Fast Statistical Analysis of Rare Failure Events for SRAM Circuits in High-Dimensional Variation Space

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Shupeng Sun

B.E., Automation, Tsinghua University

Carnegie Mellon University
Pittsburgh, PA

August, 2015

Abstract

SRAM (static random-access memory) has been widely embedded in a large amount of semiconductor chips. Therefore, the yield of most semiconductor chips is dominated by the yield of SRAM. SRAM consists of a considerable number of replicated components (e.g., SRAM bit-cell, SRAM array, sense amplifier, etc.), and the failure event for each individual component must be rare in order to achieve sufficiently high yield. Accurately estimating the rare failure rates for these replicated circuit components is a challenging task, especially when the variation space is high-dimensional. In this thesis, three novel approaches have been proposed to efficiently estimate the rare failure probability in the SRAM circuits.

First, we propose a subset simulation (SUS) technique to estimate the rare failure rates for circuit blocks which have continuous performance metrics. The key idea of SUS is to express the rare failure probability of a given circuit as the product of several large conditional probabilities by introducing a number of intermediate failure events. These conditional probabilities can be efficiently estimated with a set of Markov chain Monte Carlo samples generated by a modified Metropolis algorithm. To quantitatively assess the accuracy of SUS, a statistical methodology is further proposed to accurately estimate the confidence interval of SUS based on the theory of Markov chain Monte Carlo simulation.

Second, to efficiently estimate the rare failure rates for circuit blocks which have discrete performance metrics, scaled-sigma sampling (SSS) is proposed. SSS aims to generate random samples from a distorted probability distribution for which the standard deviation (i.e., sigma) is scaled up. Next, the failure rate is accurately estimated from these scaled random samples by using an analytical model derived from the theorem of “soft maximum”.

Finally, to further reduce the simulation cost, we propose a Bayesian scaled-sigma sampling (BSSS) approach which can be considered as an extension of SSS. The key idea of BSSS is to explore the “similarity” between different SSS models fitted at different design stages and encode it as our prior knowledge. Bayesian model fusion is then adopted to fit the SSS model with consideration of the prior

knowledge.

Acknowledgement

First and foremost, I want to thank my Ph.D. advisor Prof. Xin Li. He has always been supportive, encouraging, and inspiring during the past five years. He helped me to grow as an independent researcher. Without his invaluable guidance, I cannot make through this tough journey. His conscientiousness and infectious enthusiasm for the work has made a significant influence on me. I am forever grateful for his support and mentoring.

I would like to give special thanks to my Ph.D. thesis committee members: Prof. Shawn Blanton, Dr. Hongzhou Liu, Prof. Alexander Smola, and Dr. Ming Zhang. I want to thank you for serving as my Ph.D. thesis committee members and providing me with your priceless comments and suggestions about my thesis work. Prof. Shawn Blanton also served as a committee member in my Ph.D. qualification exam. I would like to thank Prof. Blanton for his continuous help and support during my Ph.D. program. Dr. Hongzhou Liu collaborated with us on SSS work. During the collaboration, Dr. Liu came up with many brilliant ideas, which made SSS of practical utility. Dr. Liu evaluated SSS at Cadence, and eventually integrated SSS into the Cadence commercial tool. I would like to give my sincere thanks to Dr. Liu for all his efforts to make SSS successful and influential. Prof. Alexander Smola helped and guided me to have a thorough understanding of particle filter technique. I would like to thank Prof. Smola for making my thesis more sound and complete. Dr. Zhang was my intern mentor at Samsung Semiconductor. He motivated and encouraged me to think about my research from an industrial perspective, which is extremely valuable for my thesis work. Thank you Dr. Zhang.

I would also like to thank our industry collaborators. My special thanks go to Ben Gu and Kangsheng Luo from Cadence Design Systems. My thesis work cannot be completed without their help and support.

A special acknowledgement goes to the financial support from the National Science Foundation, which makes this research possible.

My friends and office mates at Carnegie Mellon University make both my academic and personal life more interesting and enjoyable. Thank you so much.

Last but not the least, I would like to express my heartfelt appreciation to my parents. Words are not enough to express how grateful I am to them. Ph.D. life is never easy. Their unconditional love and support makes me stronger and more resolute. I love you forever and ever.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Static random-access memory (SRAM) has been widely embedded in a large amount of semiconductor chips. For example, roughly half of the area of an advanced microprocessor chip is occupied by SRAM. Therefore, the yield of most semiconductor chips is dominated by the yield of SRAM. Here, yield refers to the ratio between the number of good chips and the total number of chips. For this reason, designing a robust SRAM system with sufficiently high yield becomes an extremely important task for the IC design community [1]-[22].

SRAM typically contains a large number of replicated components (e.g., SRAM bit-cell, SRAM array, sense amplifier, etc.). Though these replicated components are designed with exactly the same parameters (e.g., transistor length and width), they can behave quite differently due to large-scale process variations. Since a failure of each individual component can cause a failure of the entire SRAM system, these replicated components must be designed to be extremely robust under process variations so that the overall yield of the SRAM system can meet the design specification. For instance, the failure rate of an SRAM bit-cell must be less than 10^{-8}~10^{-6} so that the SRAM system, containing millions of SRAM bit-cells, can achieve sufficiently high yield. To facilitate an efficient SRAM design, fast statistical methods for estimating the rare failure probability of these replicated components are strongly desired.

The simple way to estimate the failure probability is to apply the well-known brute-force Monte Carlo (MC) technique [72]. MC directly draws random samples from the probability density function (PDF) that models device-level variations, and performs a transistor-level simulation to evaluate the performance value for each random sample. Theoretically, $1/P_f$ random samples are required on average to obtain a failure sample [72]. Here, P_f denotes the failure rate. When MC is applied to estimate an extremely small failure rate (e.g., 10^{-8}~10^{-6}), a large number of (e.g., 10^7~10^9) random samples are needed to accurately estimate the small failure probability. Since generating each random sample requires a transistor-level simulation, 10^7~10^9 transistor-level simulations are needed to collect 10^7~10^9 random samples, implying that MC can be extremely expensive for our application of rare failure rate estimation.

To improve the sampling efficiency, importance sampling (IS) methods have been proposed in the literature [24]-[26], [31], [33], [35], [37], [39], [41], [44]-[45]. Instead of sampling the original PDF, IS samples a distorted PDF to get more samples in the important failure region. The efficiency achieved by IS highly depends on the choice of the distorted PDF. The traditional IS methods apply several heuristics to construct a distorted PDF that can capture the most important failure region in the variation space. Such a goal, though easy to achieve in a low-dimensional space, is extremely difficult to fulfill when a large number of random variables are used to model process variations.

Another approach to improving the sampling efficiency, called statistical blockade, has recently been proposed [34]. This approach first builds a classifier with a number of transistor-level simulations, and then draws random samples from the original PDF. Unlike MC where all the samples are evaluated by transistor-level simulations, statistical blockade only simulates the samples that are likely to fall into the failure region or close to the failure boundary based on the classifier. The efficiency achieved by this approach highly depends on the accuracy of the classifier. If the variation space is high-dimensional, a large number of transistor-level simulations are needed to build an accurate classifier, which makes this method quickly intractable.

In addition to the aforementioned statistical methods, several deterministic approaches have also been proposed to efficiently estimate the rare failure probability [28]-[29], [38], [42]. These methods first find the failure boundary, and then calculate the failure probability by integrating the PDF over the failure region in the variation space. Though efficient in the low-dimensional space, it is often computationally expensive to accurately determine the failure boundary in a high-dimensional space especially if the boundary has a complicated shape (e.g., non-convex or even discontinuous).

Most of these traditional methods [24]-[45] have been successfully applied to the SRAM cell-level design to estimate the rare failure rates for SRAM bit-cells where only a small number of (e.g., 6~20) independent random variables are used to model process variations and, hence, the corresponding variation space is low-dimensional. However, replicated circuit components, such as SRAM arrays and sense amplifiers, have hundreds or even thousands of random variables. Take an SRAM array as an example. Even a small SRAM array contains hundreds of transistors which render a high-dimensional variation space (i.e., hundreds of random variables to model the process variations). To the best of our knowledge, no traditional techniques can efficiently and accurately estimate the rare failure probability in a high-dimensional variation space. It, in turn, poses an immediate need of developing a new CAD tool to accurately capture the rare failure events in a high-dimensional variation space with low computational cost.

1.2 Thesis Contributions

To accurately capture the rare failure events in a high-dimensional variation space, a novel *subset simulation* (SUS) technique is proposed in this thesis. The key idea of SUS, borrowed from other communities [53]-[59], is to express the rare failure probability as the product of several large conditional probabilities by introducing a number of intermediate failure events. The original problem of rare failure probability estimation is then cast to an equivalent problem of estimating a sequence of conditional probabilities. Since these conditional probabilities are large, they are relatively easier to estimate than the original rare failure rate. Our experimental results in Section 3.5 show that the proposed SUS approach can accurately estimate an extremely small failure rate (e.g., 10^{-6}) even with only a few thousand simulations when hundreds of random variables are used to model the process variations. However, the traditional IS based technique fails to work.

When implementing the proposed SUS method, it is difficult, if not impossible, to directly draw random samples from the conditional PDFs and estimate the conditional probabilities, since these conditional PDFs are unknown in advance. To address this issue, a modified Metropolis (MM) algorithm is adopted from the literature [54] to generate random samples by constructing a number of Markov chains. The conditional probabilities of interest are then estimated from these random samples. Unlike most traditional techniques [24]-[45] that suffer from the dimensionality issue, SUS can be efficiently applied to high-dimensional problems, which will be demonstrated by the experimental results in Section 3.5.

It is important to emphasize that though the key idea of SUS was first proposed in other communities [53]-[59], how to estimate the confidence interval of SUS remains an open question. Due to this reason, the SUS method has not been successfully applied to many practical applications in the literature. In this work, we propose a statistical methodology to accurately estimate the confidence interval of SUS based on the theory of Markov chain Monte Carlo simulation, thereby making our proposed SUS method of practical utility.

To define the intermediate failure events required by SUS, the circuit performance of interest should be continuous. However, the performances of interest in the SRAM circuits can be discrete. Take voltage-mode sense amplifier as an example. When studying the stability of the sense amplifier, the performance of interest is the binary output of the sense amplifier. A real industry sense amplifier design contains around one hundred transistors which render a high-dimensional variation space. To estimate the rare failure probability for discrete performances of interest in a high-dimensional variation space, *scaled-sigma sampling* (SSS) is proposed. The key idea of SSS is to generate random samples from the "scaled" PDFs and estimate the "scaled" failure rates based on these random samples. Here, the "scaled" PDF refers to the distorted PDF for which the standard deviation of the original PDF is scaled up. By scaling up the standard deviation, the rare failure event associated with the original PDF becomes not-so-rare in the "scaled" PDFs and, therefore, the "scaled" failure rates corresponding to the "scaled" PDFs can be efficiently estimated.

To further recover the original rare failure rate from the scaled failure rates, we derive an analytical model between the scaled failure rate and the scaling factor based on the theorem of "soft maximum" [73], which is the key contribution of SSS work. To fit the model, we first choose a set of scaling factors, and estimate their corresponding scaled failure rates from a small number of scaled random samples. The model is then optimally fitted by applying maximum-likelihood estimation (MLE) [72]. Next, the original rare failure rate can be efficiently estimated from the fitted model by setting the scaling factor to 1.

Unlike the traditional estimator where a statistical metric is estimated by the average of multiple random samples and, hence, the confidence interval can be derived as a closed-form expression, our proposed SSS estimator is calculated by linear regression with nonlinear exponential/logarithmic transformation, as shown in Chapter 4. Therefore, accurately estimating the confidence interval of SSS is not a trivial task. To address the aforementioned challenge, we apply bootstrap technique [70]. The key idea of bootstrap is to re-generate a large number of random samples based on a statistical model without running additional transistor-level simulations. These random samples are then used to repeatedly calculate the failure rate by SSS for multiple times. Based on these repeated runs, the statistics (hence, the confidence interval) of the proposed SSS estimator can be accurately estimated.

Though only a few thousand simulations are required by SUS and SSS for rare failure probability estimation as demonstrated in Section 3.5 and 4.4, tens of thousands of simulations in total may be needed when designing a circuit component. To understand the reason, let us assume that we are designing a sense amplifier, and applying SSS to estimate the failure probability. If the estimated failure probability meets our design specification, the design process is complete. Otherwise, we need to improve our sense amplifier design, and re-run SSS. Generally speaking, we need a few iterations before we converge to the final design. Since SSS spends a few thousand simulations for each design candidate, tens of thousands of simulations may be required over the entire design process, which can be extremely expensive.

To further reduce the simulation cost, we propose a novel Bayesian scaled-sigma sampling (BSSS) approach which can be considered as an extension of SSS. SSS is a model-based approach. By studying SSS, we can observe that a number of coefficients of the SSS model for an early design candidate are similar to those for a late design candidate. Motivated by this observation, we propose to explore the "similarity" between different SSS models fitted for different design candidates, and encode such "similarity" as our prior distribution for the SSS model coefficients. Next, we apply Bayesian model fusion (BMF) [60]-[69] to fit the SSS model with the prior.

The key difference between BSSS and SSS lies in the fact that BSSS maximizes the product of the prior distribution and the data likelihood by maximum-a-posteriori (MAP) estimation [72], while SSS only maximizes the data likelihood by MLE. As long as the prior distribution is properly defined, MAP can reduce the amount of required simulation data and, hence, the model fitting cost without surrendering any accuracy, as demonstrated in [60]-[69]. In other words, BSSS can be more efficient than SSS if the prior distribution is appropriately defined. Similar to SSS, we apply bootstrap technique [70] to accurately estimate the confidence interval of our BSSS estimator. Our experimental results in Section 5.5 demonstrate that BSSS can achieve superior accuracy over SSS when the prior distribution is appropriately defined.

1.3 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we first introduce background knowledge relevant to SRAM rare failure probability estimation. Next, several state-of-the-art methodologies for rare failure rate estimation in the CAD community are reviewed, and their algorithm limitations are discussed.

In Chapter 3, our proposed SUS approach is described. We start with a simple 2-dimensional example to illustrate the key idea of SUS. Next, the SUS idea is extended to the general high-dimensional scenario. How to select the subsets and estimate the confidence interval for our proposed SUS estimator are carefully studied. Numerical experimental results of an SRAM column example are used to demonstrate the efficacy of SUS. Algorithm limitations of SUS are discussed at the end of the chapter.

In Chapter 4, we present our proposed SSS approach. First, we describe SSS for Gaussian random variables. Next, SSS is extended to handle Gaussian and uniform random variables. Several implementation issues are then discussed in detail, including (1) model fitting via MLE, (2) confidence interval estimation via bootstrap, and (3) scaling factor selection. Two circuit examples are used to demonstrate the efficacy of SSS. Algorithm limitations of SSS are discussed to complete the chapter.

In Chapter 5, we describe our proposed BSSS approach. First, we talk about the motivation of this work. Next, we present an important observation about SSS and then introduce the key idea of BSSS based on this observation. Implementation issues including (1) prior definition and (2) model fitting via BMF, are then discussed in detail. Finally, a voltage-mode sense amplifier example is used to demonstrate the efficacy of BSSS.

Chapter 6 concludes the thesis. Several potential future research directions relevant to this work are discussed.

Chapter 2

Background

2.1 Static Random-Access Memory (SRAM)

The simplified SRAM architecture is shown in Figure 2-1. Blue lines denote bit lines, and red lines denote word lines. SRAM is typically composed of SRAM bit-cells, sense amplifiers (SAs), pre-charge circuits, cell I/O circuits, address decoder, etc. [74].

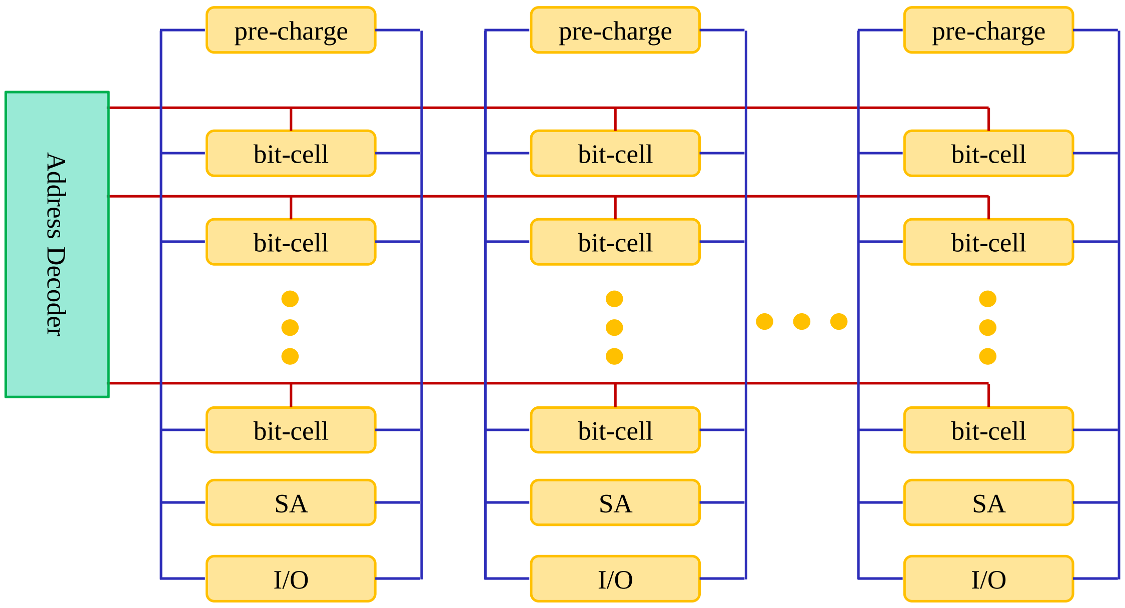

Figure 2-1. Simplified SRAM architecture.

SRAM bit-cell is the fundamental component of SRAM system, and stores binary value (0 or 1) when it is powered. There are several different SRAM bit-cell designs (e.g., 6-transistor design, 8-transistor design, etc.), and each of them has its own advantages and disadvantages regarding various design specifications (e.g., cell density, speed, stability, etc.). Here, we take the well-known 6-transistor (6T)

SRAM bit-cell design as an example to illustrate the basic SRAM operations (e.g., read and write). The schematic of the 6T SRAM bit-cell design is shown in Figure 2-2, where *WL* denotes word line, and *BL* and *BLB* denote two bit lines. 6T SRAM bit-cell design is a symmetric design. Namely, two pull-down transistors (i.e., M_1 and M_2), two pass-gate transistors (i.e., M_3 and M_4), and two pull-up transistors (i.e., M_5 and M_6) are identical respectively. M_1, M_2, M_5, M_6 form two cross-coupled inverters, resulting in two stable states. More specifically, if V_Q (i.e., the voltage of node *Q*) is 0, V_{QB} (i.e., the voltage of node *QB*) should be V_{DD} due to the positive feedback of two cross-coupled inverters. Otherwise, if V_Q is V_{DD}, V_{QB} should be 0. These two stable states are used to represent binary values (0 or 1) in SRAM circuits. Without loss of generality, we assume that bit-cell stores 0 if V_Q is 0 and stores 1 if V_Q is V_{DD}. Two pass-gate transistors (i.e., M_3 and M_4) are used to control the bit-cell accessibility. Once the *WL* voltage is high, M_3 and M_4 are activated, and bit-cell can be read and written. Once the *WL* voltage is low, M_3 and M_4 are turned off, and bit-cell preserves its state and stays in the standby mode.

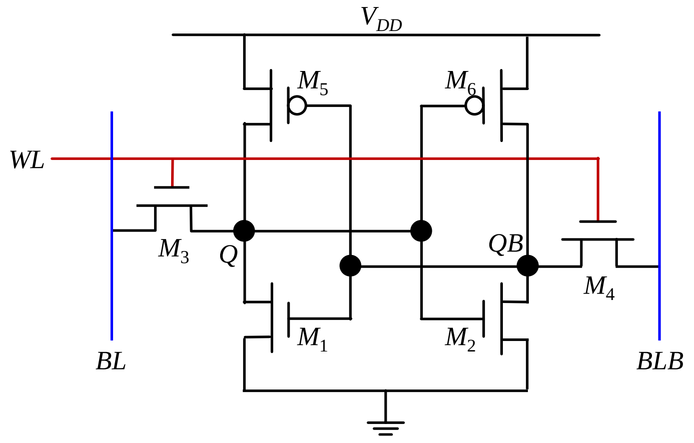

Figure 2-2. 6-transistor SRAM bit-cell schematic.

SRAM has three modes: (1) reading, (2) writing, and (3) standby. Without losing generality, we assume that the SRAM bit-cell shown in Figure 2-2 stores 0. Namely, V_Q is 0, and V_{QB} is V_{DD}. The three modes can be simply described as follows:

- **Reading**: two bit lines (i.e., *BL* and *BLB* in Figure 2-2) are first pre-charged to the supply voltage V_{DD} by pre-charge circuits. Next, word line (i.e., *WL* in Figure 2-2) is activated, and two pass-gate transistors are turned on. Since V_Q is 0, there is current flowing from *BL* to *Q*, which decreases the *BL* voltage. On the other hand, both *BLB* and *QB* have high voltages and, hence, the *BLB* voltage is almost unchanged. In a word, *BL* and *BLB* have different discharging rates, resulting in a significant voltage difference after activating *WL* for certain time. By sensing the voltage difference between *BL* and *BLB*, we can tell the value stored in the SRAM bit-cell. If *BL* voltage is smaller than *BLB* voltage, bit-cell stores 0. Otherwise, if *BL* voltage is larger than *BLB* voltage, bit-cell stores 1.

- **Writing**: assume that we want to write 1 into the cell. Since the cell currently stores 0, we aim to change V_Q from 0 to V_{DD}, and V_{QB} from V_{DD} to 0. Towards this end, we activate *WL* to turn on two pass-gate transistors. The *BL* and *BLB* voltages are driven to *VDD* and 0 respectively by the writing circuits. If the driving power of M_4 is stronger than that of M_6, V_{QB} decreases. Once V_{QB} drops below the threshold voltage of M_1, M_1 is turned off and V_Q starts to increase. Once V_Q rises above the threshold voltage of M_2, M_2 is turned on, making *QB* discharged more quickly. V_{QB} keeps decreasing and V_Q keeps increasing. Eventually, M_5 is turned on and M_6 is turned off. V_{QB} becomes 0, and V_Q becomes V_{DD}.
- **Standby**: during the standby mode, *WL* is not activated, which isolates bit-cell from all other circuits. Bit-cell preserves its value.

To function correctly, three types of transistors (i.e., pull-down, pull-up and pass-gate) must be carefully sized. Generally speaking, pull-down transistor should be stronger than pass-gate transistor to guarantee read stability, and pass-gate transistor should be stronger than pull-up transistor to insure successful writes [74].

Although SRAM system is carefully designed and guaranteed to function correctly at the nominal process corner, failures can occur due to imperfect manufacturing process. Briefly speaking, there are two kinds of failures caused by manufacturing process variations:

- **Catastrophic failure**: failures caused by circuit shorts and opens. E.g., if *Q* is short to ground in Figure 2-2, the bit-cell can only store 0. If we want to write 1 into such cell, we get write failures.
- **Parametric failure**: parametric variations [1]-[23] change the driving capability of each device, and can cause several different kinds of failures:

    1) **Read access failure**: during read operation, the voltage difference between two bit lines increases. If the voltage difference is not large enough after pre-defined reading time, the value stored in bit-cell may not be correctly detected by sense amplifier (SA), resulting in read access

failure. Take the bit-cell shown in Figure 2-2 as an example. V_Q is 0, and V_{QB} is V_{DD}. If M_1 becomes weak (e.g., transistor width decreases) after manufacturing, the current flowing from *BL* to *Q* becomes smaller than that at the nominal condition. If the current is too small, the voltage difference between *BL* and *BLB* may be negligible, leading to a read access failure.

2) **Read stability failure**: there is current flowing from *BL* to *Q* during read operation for the bit-cell shown in Figure 2-2. If pull-down transistor M_1 becomes weak (e.g., the transistor width decreases) and pass-gate transistor M_3 becomes strong (e.g., the transistor width increases) after manufacturing, V_Q can be significantly larger than 0 due to the resistive voltage division between M_1 and M_3. If V_Q is larger than the threshold voltage of M_2, the bit-cell may flip, leading to a read stability failure.

3) **Write failure**: ideally, pass-gate transistor should be stronger than pull-up transistor in order to achieve successful writes. After manufacturing, however, pull-up transistors could be stronger than pass-gate transistors. If this occurs, it is very likely that the desired value cannot be written into the bit-cell, resulting in a write failure.

Catastrophic failure and parametric failure can be estimated independently. In this thesis, we only focus on parametric failure rate estimation. Mathematical formulation for parametric failure rate estimation is described in the following sub-section.

2.2 SRAM Yield Estimation

Since the yield of most semiconductor chips is dominated by the yield of SRAM, it is extremely important to estimate SRAM production yield before manufacturing. In this sub-section, we describe the mathematical formulation for SRAM parametric yield (or failure rate) estimation.

Parametric failures are caused by parametric variations, which refer to the deviations between manufactured device-level parameter values and designed device-level parameter values [23]. These device-level parameters include transistor length (*L*), transistor width (*W*), threshold voltage (V_{th}), oxide thickness (T_{OX}), mobility (μ), etc. Due to parametric variations, even identically designed devices can behave quite differently. Parametric variations can be categorized into local mismatches and global variations. Local mismatches between different devices can be considered as independent. Global variations, however, are strongly correlated between different devices on the chip. To model these two different variations, process design kit (PDK) uses two different kinds of variables: (1) local mismatch random variable and (2) global random variable.

Since local mismatches between different devices are independent, PDK uses different sets of local mismatch random variables to model local mismatches for different devices. Without losing generality, we assume that M_L independent local mismatch random variables are used to model local mismatches for each device:

$$\mathbf{x}_{Local} = \begin{bmatrix} x_{Local,1} & x_{Local,2} & \cdots & x_{Local,M_L} \end{bmatrix}. \tag{2.1}$$

If we have D devices, $M_L \times D$ independent random variables are used to model local mismatches:

$$\begin{bmatrix} \mathbf{x}_{Local}^{(1)} & \cdots & \mathbf{x}_{Local}^{(D)} \end{bmatrix} = \begin{bmatrix} x_{Local,1}^{(1)} & \cdots & x_{Local,M_L}^{(1)} & \cdots & x_{Local,1}^{(D)} & \cdots & x_{Local,M_L}^{(D)} \end{bmatrix}. \tag{2.2}$$

We further assume that M_G independent global random variables are used to model global variations for all the devices:

$$\mathbf{x}_{Global} = \begin{bmatrix} x_{Global,1} & x_{Global,2} & \cdots & x_{Global,M_G} \end{bmatrix}. \tag{2.3}$$

Given (2.1) and (2.3), the deviations of transistor device-level parameters can be written as functions of these M_L local mismatch random variables and M_G global random variables:

$$\begin{aligned} \Delta L &= function\left(\mathbf{x}_{Local}, \mathbf{x}_{Global}\right) \\ \Delta W &= function\left(\mathbf{x}_{Local}, \mathbf{x}_{Global}\right) \\ \Delta V_{th} &= function\left(\mathbf{x}_{Local}, \mathbf{x}_{Global}\right) \\ &\vdots \end{aligned} \tag{2.4}$$

If a circuit has D transistors, the deviation of our interested performance metric (say, y) can be written as:

$$\Delta y = function\left[\underbrace{\Delta L^{(1)}, \Delta W^{(1)}, \Delta V_{th}^{(1)}, \cdots}_{\text{1st transistor}}, \underbrace{\Delta L^{(2)}, \Delta W^{(2)}, \Delta V_{th}^{(2)}, \cdots}_{\text{2nd transistor}}, \cdots, \underbrace{\Delta L^{(D)}, \Delta W^{(D)}, \Delta V_{th}^{(D)}, \cdots}_{D\text{th transistor}}\right]. \tag{2.5}$$

Given (2.4), Eq. (2.5) can be re-written as:

$$\Delta y = function\left[\mathbf{x}_{Local}^{(1)}, \mathbf{x}_{Local}^{(2)}, \cdots, \mathbf{x}_{Local}^{(D)}, \mathbf{x}_{Global}\right]. \tag{2.6}$$

Note that all the transistors share the same set of global random variables and, therefore, there is only a single set of global random variables in the parameter list of (2.6).

A straightforward way to estimate SRAM yield is to generate a number of (say, N) samples $\{\mathbf{x}^{(n)}; n = 1, 2, \cdots, N\}$ according to the statistical models defined for $\mathbf{x}_{Local}$ and $\mathbf{x}_{Global}$ in PDK. If SRAM system has D transistors in total, each sample $\mathbf{x}$ is composed of $M_L \times D + M_G$ random variables:

$$\mathbf{x} = \Big[\underbrace{\mathbf{x}_{Local}^{(1)}}_{M_L \text{ variables}}, \underbrace{\mathbf{x}_{Local}^{(2)}}_{M_L \text{ variables}}, \cdots, \underbrace{\mathbf{x}_{Local}^{(D)}}_{M_L \text{ variables}}, \underbrace{\mathbf{x}_{Global}}_{M_G \text{ variables}} \Big]. \tag{2.7}$$

For each sample $\mathbf{x}^{(n)}$ where $n \in \{1, 2, \cdots, N\}$, we run a transistor-level simulation to evaluate its corresponding performance metric $y^{(n)}$. Based on $\{y^{(n)}; n = 1, 2, \cdots, N\}$, we can estimate SRAM yield by using brute-force Monte Carlo approach which will be described in Section 2.3.1. This approach, though easy to implement, has two limitations:

1) SRAM system has millions or even billions of nodes. Running a transistor-level simulation for the entire SRAM system is extremely challenging, if not impossible.
2) If the estimated SRAM yield does not meet our specification, we need to tune our design. Unfortunately, we do not know how to tune the circuit with only an estimated yield value. In other words, we do not know which part or circuit component causes the failure based on a single yield value.

Due to these limitations, this approach is not applied in practice.

SRAM system contains a large number of identically designed circuit components (e.g., SRAM column, sense amplifier, etc.). We can take advantage of such property and have a more efficient and practical approach to estimate SRAM system yield. Before describing this practical approach, let us use a simple example to illustrate the key idea of this approach. Here, we assume that

1) There is no error-correcting code or redundancy circuits [74].
2) Only SRAM columns have failures and all other circuit components (e.g., address decoder) function correctly under large-scale process variations.
3) There are R SRAM columns and all of them are identically designed.

Since an SRAM column has only a few hundred nodes, running a transistor-level simulation for an SRAM column is affordable. Therefore, we can afford a reasonable number of (e.g., a few thousand) transistor-level simulations for SRAM column failure rate estimation. However, estimating the failure rate for a single SRAM column is still nontrivial. An SRAM column is composed of a number of (e.g., 64)

SRAM bit-cells, and the failure for any SRAM bit-cell leads to an SRAM column failure. Due to this reason, to check whether an SRAM column fails, we need to check all the SRAM bit-cells inside the column, which requires intensive transistor-level simulations. How to efficiently estimate the SRAM column failure rate is still an open question. In this thesis, we derive a conservative upper bound for the SRAM column failure rate based on the SRAM bit-cell failure rate, as shown in Section 3.5. Unlike most traditional approaches that only consider the bit-cell itself when estimating the SRAM bit-cell failure rate, we consider the entire SRAM column. By considering the entire SRAM column, we can estimate the SRAM bit-cell failure rate more accurately, which will be explained later. Here, we use P_f to denote the failure rate of a single SRAM column, and P_f can be written as

$$P_f = function\left[\mathbf{x}_{Local}^{(column)}, \mathbf{x}_{Global}\right] \tag{2.8}$$

where $\mathbf{x}_{Local}{}^{(column)}$ includes all the local mismatch random variables for a single SRAM column, and $\mathbf{x}_{Global}$ includes all the global random variables. Since all SRAM columns are identically designed, their failure rates are the same. Note that our goal is to estimate SRAM system yield. The question we need to answer here is that how to calculate SRAM system yield based on failure rates of these SRAM columns.

If P_f is estimated by varying both $\mathbf{x}_{Local}{}^{(column)}$ and $\mathbf{x}_{Global}$, failures between different SRAM columns are correlated due to the fact that they share the same set of global random variables $\mathbf{x}_{Global}$. If failures are correlated, it is extremely difficult to calculate the overall failure rate of all SRAM columns. To remove the correlation, we can set $\mathbf{x}_{Global}$ to fixed values, and estimate P_f by only varying $\mathbf{x}_{Local}{}^{(column)}$. As failures between different SRAM columns become conditionally independent by setting $\mathbf{x}_{Global}$ to fixed values, the overall failure rate of all SRAM columns can be calculated by

$$P_{all} = 1 - \left(1 - P_f\right)^R \tag{2.9}$$

where P_{all} denotes the overall failure rate of all SRAM columns.

We can generate a number of (say, C) samples $\{\mathbf{x}_{Global}{}^{(c)}; c = 1, 2, \cdots, C\}$ according to the statistical model defined for $\mathbf{x}_{Global}$ in PDK. For each sample $\mathbf{x}_{Global}{}^{(c)}$ where $c \in \{1, 2, \cdots, C\}$, we estimate the failure rate for a single SRAM column given that $\mathbf{x}_{Global} = \mathbf{x}_{Global}{}^{(c)}$, and calculate the overall failure rate $P_{all}{}^{(c)}$ by (2.9). Last, we estimate SRAM system yield by:

$$yield = 1 - \frac{1}{C}\sum_{c=1}^{C} P_{all}^{(c)} . \tag{2.10}$$

From (2.9) and (2.10), we observe that to achieve sufficiently high SRAM production yield, P_f must be extremely small. To understand this, let us assume that $R = 10^4$. If $P_f = 10^{-4}$, $P_{all} \approx 0.63$ which is too large. If $P_f = 10^{-5}$, $P_{all} \approx 0.10$ which is much better than 0.63. From these numbers, we can see that the failure rate of each SRAM column must be smaller than 10^{-5} in this simplified SRAM system in order to achieve a robust SRAM design.

Now let us describe this approach in a more general way. Suppose that we partition SRAM into R components, including SRAM columns, sense amplifiers, address decoder, cell I/O, etc. There are several different ways to partition an SRAM system. For instance, we can consider an SRAM column as a single component. Another way to partition an SRAM column is to consider each SRAM bit-cell inside the column as a single component and, hence, we have a number of small components in the column. A valid partition should meet the following requirements:

1) Different circuit components do not share any devices.
2) Failures of different circuit components should be weakly correlated given that global variables are fixed.

Given a valid partition, Algorithm 2.1 describes the SRAM system yield estimation flow.

Algorithm 2.1 SRAM Yield Estimation Flow

```
1.  Generate $C$ samples $\{\mathbf{x}_{Global}^{(c)}; c = 1, 2, \cdots, C\}$ according to the statistical model defined for $\mathbf{x}_{Global}$ in PDK
2.  For $c = 1, 2, \cdots, C$
3.      Set $\mathbf{x}_{Global} = \mathbf{x}_{Global}^{(c)}$
4.      For $r = 1, 2, \cdots, R$
5.          If the $h$-th circuit component has the same design as the $r$-th circuit component where $h \in \{1, 2, \cdots, r-1\}$, set $P_f^{(r)} = P_f^{(h)}$. Otherwise, estimate failure rate $P_f^{(r)}$ for the $r$-th circuit component
6.      End For
7.      Calculate the overall failure rate by
```

$$P_{all}^{(c)} = 1 - \prod_{r=1}^{R} \left[1 - P_f^{(r)} \right] \tag{2.11}$$

```
8.  End For
9.  Calculate SRAM system yield by (2.10)
```

There are two important clarifications that we should make for Algorithm 2.1. First, Eq. (2.11) holds given the assumption that failures of different circuit components are weakly correlated when global variables are fixed. To achieve an accurate estimation by Algorithm 2.1, we should minimize correlations between different circuit components when we partition SRAM. Most traditional approaches typically consider each SRAM bit-cell as a single circuit component. Since simulating each SRAM bit-cell consumes a very short time, the traditional partition makes SRAM yield estimation pretty efficient. However, such partition induces non-negligible correlations since different bit-cells inside each SRAM column share the same bit-line and peripheral circuits, such as sense amplifier. Therefore, the traditional partition could lead to a significant accuracy loss. To address this issue, we consider each SRAM column as an individual circuit component in this thesis.

Second, we need to emphasize that it is extremely difficult to obtain a partition where all the circuit components are weakly correlated. If a valid partition cannot be achieved, we can slightly modify Algorithm 2.1 and compute a conservative lower bound for the SRAM system yield. To this end, we only need to replace (2.11) at Step 7 with the following equation:

$$P_{all}^{(c)} \approx \sum_{r=1}^{R} P_f^{(r)} \ . \tag{2.12}$$

The right hand side of (2.12) is a conservative upper bound for $P_{all}^{(c)}$. Therefore, by replacing (2.11) with (2.12) in Algorithm 2.1, we obtain a conservative lower bound for the SRAM system yield.

From Algorithm 2.1, we observe that estimating failure rates for individual circuit components in Step 5 is executed many times. Hence, efficient failure rate estimation for these small individual circuit components is extremely important for SRAM system yield estimation.

Some circuit components appear tens of thousands of times in a typical SRAM system, such as SRAM column, sense amplifier, etc. To achieve sufficiently high SRAM production yield, failure rates for these replicated circuit components must be extremely small, as explained earlier in this sub-section. In a word, *efficient rare* failure rate estimation approaches for individual circuit components when global variables are fixed are highly desired. In this thesis, we will focus on analyzing rare circuit failure events in the presence of local mismatches only.

Without loss of generality, suppose that there are D transistors in our interested circuit block and the M-dimensional vector $\mathbf{x}$ includes all the independent local mismatch random variables for these D transistors

$$\mathbf{x} = \left[\underbrace{x_1 \ \cdots \ x_{M_L}}_{\text{1st transistor}} \quad \underbrace{x_{M_L+1} \ \cdots \ x_{2\times M_L}}_{\text{2nd transistor}} \quad \cdots \quad \underbrace{x_{(D-1)\times M_L+1} \ \cdots \ x_{D\times M_L}}_{D\text{th transistor}} \right]^T . \tag{2.13}$$

Here, $M = D\times M_L$. Our interested circuit block could have a few hundred transistors, such as SRAM columns. Hence, M could easily reach a few hundred or even a few thousand, rendering a high-dimensional variation space. The failure rate P_f of a circuit can be mathematically represented as

$$P_f = \int_{\Omega} f(\mathbf{x}) \cdot d\mathbf{x} \tag{2.14}$$

where $f(\mathbf{x})$ denotes the probability density function (PDF) for $\mathbf{x}$, and Ω denotes the failure region in the variation space, i.e., the subset of the variation space where the performance of interest does not meet the specification. Alternatively, the failure rate in (2.14) can be defined as

$$P_f = \int_{-\infty}^{+\infty} I(\mathbf{x}) \cdot f(\mathbf{x}) \cdot d\mathbf{x} \tag{2.15}$$

where $I(\mathbf{x})$ represents the indicator function

$$I(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in \Omega \\ 0 & \mathbf{x} \notin \Omega \end{cases} . \tag{2.16}$$

In the following sub-sections, we will introduce several traditional approaches for rare failure rate estimation used in the CAD community and highlight their limitations before ending this background section.

2.3 Traditional Approaches

2.3.1 Brute-Force Monte Carlo

The failure rate P_f can be estimated by brute-force Monte Carlo (MC) analysis. The key idea of MC is to draw N random samples $\{\mathbf{x}^{(n)}; n = 1, 2, \cdots, N\}$ from $f(\mathbf{x})$, and perform transistor-level simulations to evaluate their performance values. The failure rate P_f is then estimated by [72]

$$P_f^{MC} = \frac{1}{N} \cdot \sum_{n=1}^{N} I\left[\mathbf{x}^{(n)}\right] \tag{2.17}$$

where $I(\mathbf{x})$ is defined in (2.16). The variance of the estimator P_f^{MC} in (2.17) can be approximated as [72]

$$v_f^{MC} = \frac{P_f^{MC} \cdot \left(1 - P_f^{MC}\right)}{N}. \tag{2.18}$$

When MC is applied to estimate P_f that is extremely small (e.g., 10^{-8}~10^{-6}), most random samples drawn from the PDF $f(\mathbf{x})$ do not fall into the failure region Ω. Therefore, a large number of (e.g., 10^7~10^9) samples are needed to accurately estimate P_f. Note that each Monte Carlo sample is created by running an expensive transistor-level simulation. In other words, 10^7~10^9 simulations must be performed in order to collect 10^7~10^9 samples. It, in turn, implies that MC can be extremely expensive for our application of rare failure rate estimation.

2.3.2 Importance Sampling

To reduce the computational cost, several statistical algorithms based on importance sampling have been proposed [24]-[26], [31], [33], [35], [37], [39], [41], [44]-[45]. The key idea of importance sampling is to sample a distorted PDF $g(\mathbf{x})$, instead of the original PDF $f(\mathbf{x})$, so that most random samples fall into the failure region Ω. Here, $g(\mathbf{x})$ is positive for all $\mathbf{x}$ in Ω. In this case, the failure rate can be expressed as

$$P_f = \int_{-\infty}^{+\infty} \frac{I(\mathbf{x}) \cdot f(\mathbf{x})}{g(\mathbf{x})} \cdot g(\mathbf{x}) \cdot d\mathbf{x}. \tag{2.19}$$

If N random samples $\{\mathbf{x}^{(n)}; n = 1, 2, \cdots, N\}$ are drawn from $g(\mathbf{x})$, the failure rate in (2.19) can be approximated by

$$P_f^{IS} = \frac{1}{N} \cdot \sum_{n=1}^{N} \frac{f\left[\mathbf{x}^{(n)}\right] \cdot I\left[\mathbf{x}^{(n)}\right]}{g\left[\mathbf{x}^{(n)}\right]}. \tag{2.20}$$

The variance of the estimator P_f^{IS} in (2.20) can be approximated as [72]

$$v_f^{IS} = \frac{1}{(N-1)\cdot N}\cdot\sum_{n=1}^{N}\left(\frac{f\left[\mathbf{x}^{(n)}\right]\cdot I\left[\mathbf{x}^{(n)}\right]}{g\left[\mathbf{x}^{(n)}\right]} - P_f^{IS}\right)^2 . \tag{2.21}$$

When a finite number of samples are available, the results from (2.17) and (2.20) can be substantially different. If $g(\mathbf{x})$ is properly chosen for importance sampling, P_f^{IS} in (2.20) can be much more accurate than P_f^{MC} in (2.17). Theoretically, the optimal $g(\mathbf{x})$ leading to maximum estimation accuracy is defined as [72]

$$g^{OPT}(\mathbf{x}) = \frac{f(\mathbf{x})\cdot I(\mathbf{x})}{P_f} . \tag{2.22}$$

Intuitively, if $g^{OPT}(\mathbf{x})$ is applied, P_f^{IS} in (2.20) becomes a constant with zero variance. Therefore, the failure rate can be accurately estimated by P_f^{IS} with very few samples.

Eq. (2.22) implies that the optimal PDF $g^{OPT}(\mathbf{x})$ is non-zero if and only if the variable $\mathbf{x}$ sits in the failure region. Namely, we should directly sample the failure region to achieve maximum accuracy. Furthermore, $g^{OPT}(\mathbf{x})$ is proportional to the original PDF $f(\mathbf{x})$. In other words, the entire failure region should not be sampled uniformly. Instead, we should sample the high-probability failure region that is most likely to occur.

Applying importance sampling, however, is not trivial in practice. The optimal PDF $g^{OPT}(\mathbf{x})$ in (2.22) cannot be easily found, since the indicator function $I(\mathbf{x})$ is unknown. Most existing importance sampling methods attempt to approximate $g^{OPT}(\mathbf{x})$ by applying various heuristics. The key idea is to first search the high-probability failure region and then a distorted PDF $g(\mathbf{x})$ is constructed to directly draw random samples from such a high-probability failure region. In the next few sub-sections, we will introduce a number of state-of-the-art importance sampling based approaches in the CAD community.

2.3.2.1 Mixture Importance Sampling

Mixture importance sampling (MIS) assumes that the vector $\mathbf{x}$ in (2.13) follows a multivariate Normal distribution [26]. Without loss of generality, we further assume that the random variables $\{x_m; m = 1, 2, \cdots, M\}$ in the vector $\mathbf{x}$ are mutually independent and follow standard Normal distributions

$$f\left(\mathbf{x}\right)=\prod_{m=1}^{M}\left[\frac{1}{\sqrt{2\pi}}\cdot\exp\left(-\frac{x_m^2}{2}\right)\right]=\frac{\exp\left(-\|\mathbf{x}\|_2^2/2\right)}{\left(\sqrt{2\pi}\right)^M} \tag{2.23}$$

where $\|\bullet\|_2$ denotes the L2-norm of a vector. Any correlated random variables that are jointly Normal can be transformed to the independent Normal random variables $\{x_m;\ m = 1, 2, \cdots, M\}$ by principal component analysis [72].

Given $f(\mathbf{x})$ in (2.23), MIS applies a two-stage flow to estimate rare failure rates:

Algorithm 2.2 Mixture Importance Sampling

1. Generate N_1 random samples from a uniform distribution $U(\mathbf{x})$. Calculate the gravity center of all failure samples. Denote the gravity center as $\mathbf{x}_c$.
2. Construct a distorted distribution $g(\mathbf{x})$

$$g\left(\mathbf{x}\right)=f\left(\mathbf{x}-\mathbf{x}_c\right)=\frac{\exp\left(-\|\mathbf{x}-\mathbf{x}_c\|_2^2/2\right)}{\left(\sqrt{2\pi}\right)^M} \tag{2.24}$$

   where the shifted mean $\mathbf{x}_c$ is obtained from Stage 1. Generate N_2 random samples from $g(\mathbf{x})$ and estimate failure rate by (2.20).

MIS described in Algorithm 2.2 generates N_1 random samples to find the shifted mean for the distorted Normal distribution $g(\mathbf{x})$, and N_2 random samples to perform importance sampling. In total, MIS runs (N_1+N_2) transistor-level simulations to estimate failure rates. To clearly understand MIS, let us use a simple 1-dimensional example to illustrate Algorithm 2.2.

As shown in Figure 2-3 (a), a number of random samples are first drawn from a uniform distribution $U(\mathbf{x})$. Here, yellow points denote samples that do not belong to the failure region Ω (i.e., the red color region), and green points denote samples that belong to the failure region Ω. Given all the green points, MIS calculates their gravity center:

$$\mathbf{x}_c=\sum_i w_i\cdot\mathbf{x}_{f,i} \tag{2.25}$$

where $\mathbf{x}_{f,i}$ denotes the i-th green point in Figure 2-3 (a), and w_i denotes the weight assigned for $\mathbf{x}_{f,i}$. Typically, if a green point is closer to the origin $\mathbf{0}$, it has a relatively larger weight. By defining weights in this way, it is very likely that the gravity center $\mathbf{x}_c$ is close to the failure boundary denoted by the red dashed line in Figure 2-3 and, hence, sits in the high-probability failure region since $f(\mathbf{x})$ follows a standard Normal distribution.

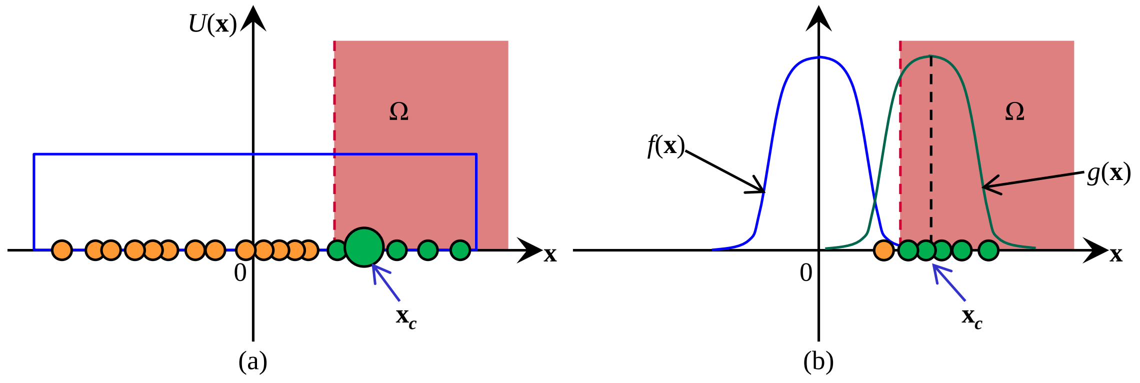

Figure 2-3. A simple 1-dimensional example is used to illustrate the key idea of MIS described in Algorithm 2.2. Yellow points denote samples that do not belong to the failure region Ω, and green points denote samples that belong to the failure region Ω. (a) Generate random samples from a uniform distribution $U(\mathbf{x})$, and calculate the gravity center of all failure samples (i.e., green points). (b) Construct a shifted Normal distribution $g(\mathbf{x})$, and generate random samples from $g(\mathbf{x})$. Calculate failure rate with these random samples by using importance sampling equation (2.20).

Having $\mathbf{x}_c$, MIS constructs a shifted Normal distribution $g(\mathbf{x})$, as shown in Figure 2-3 (b). Next, MIS generates a number of random samples from $g(\mathbf{x})$, and estimates failure rate based on these random samples by using (2.20). If the gravity center $\mathbf{x}_c$ sits in the high-probability failure region, most samples drawn from $g(\mathbf{x})$ fall into the high-probability failure region. Therefore, MIS can achieve superior estimation accuracy compared to MC.

If we have a large number of random variables in the vector $\mathbf{x}$, most samples drawn from the uniform distribution $U(\mathbf{x})$ in Stage 1 of Algorithm 2.2 are far away from the origin (i.e., $\mathbf{x} = \mathbf{0}$). Due to this reason, we may not have failure samples that fall into the high-probability failure region given that N_1 is small and, hence, the gravity center could sit in the low-probability failure region in the high-dimensional variation space. If this occurs, most samples drawn from the shifted Normal distribution $g(\mathbf{x})$ do not fall into the high-probability failure region and, therefore, MIS could be inefficient. In a word, we may not be able to efficiently apply MIS in a high-dimensional variation space.

2.3.2.2 Minimum-Norm Importance Sampling

Minimum-norm importance sampling (MNIS) also assumes that the vector **x** in (2.13) follows a multivariate Normal distribution defined in (2.23), and applies a two-stage flow to estimate rare failure rates [37]

Algorithm 2.3 Minimum-Norm Importance Sampling

1. Try to find a failure sample that is closest to the origin (i.e., $\mathbf{x} = \mathbf{0}$) with N_1 random samples. Denote this sample as $\mathbf{x}_c$.
2. Construct a distorted distribution $g(\mathbf{x})$ in (2.24) where the shifted mean $\mathbf{x}_c$ is obtained from Stage 1. Generate N_2 random samples from $g(\mathbf{x})$ and estimate failure rate by (2.20).

To clearly understand MINS described in Algorithm 2.3, let us consider a simple 1-dimensional example. As shown in Figure 2-4 (a), MNIS tries to find the failure sample that is closest to the origin $\mathbf{x} = \mathbf{0}$ at Stage 1 of Algorithm 2.3. Note that it is extremely difficult, if not impossible, to find the minimum-norm failure point. MNIS tries to find a failure point that is very close to the minimum-norm failure point by using a searching algorithm. Denote such approximated minimum-norm failure point as $\mathbf{x}_c$. Next, MNIS constructs a shifted Normal distribution $g(\mathbf{x})$ with $\mathbf{x}_c$, and performs importance sampling with $g(\mathbf{x})$ as shown in Figure 2-4 (b).

Both MNIS described in Algorithm 2.3 and MIS described in Algorithm 2.2 apply a two-stage flow to estimate rare failure rates. In the first stage, they determine a failure sample $\mathbf{x}_c$. Next, they construct a shifted Normal distribution $g(\mathbf{x})$ with $\mathbf{x}_c$ in the second stage and perform importance sampling with the shifted Normal distribution $g(\mathbf{x})$. The key difference between MNIS and MIS is how they determine $\mathbf{x}_c$ in the first stage. MIS considers the gravity center of a bunch of failure samples drawn from a uniform distribution as $\mathbf{x}_c$, while MNIS considers an approximated minimum-norm failure sample as $\mathbf{x}_c$. There are two reasons explaining why MNIS chooses the approximated minimum-norm failure sample as the shifted mean: (1) high-probability failure region is closer to the origin than the low-probability failure region under the assumption that $f(\mathbf{x})$ is defined in (2.23), and (2) we should sample the high-probability failure region in order to achieve maximum accuracy when applying importance sampling technique. Therefore, by selecting the minimum-norm failure sample as the shifted mean, most samples drawn from the shifted Normal distribution are likely to fall into the high-probability failure region.

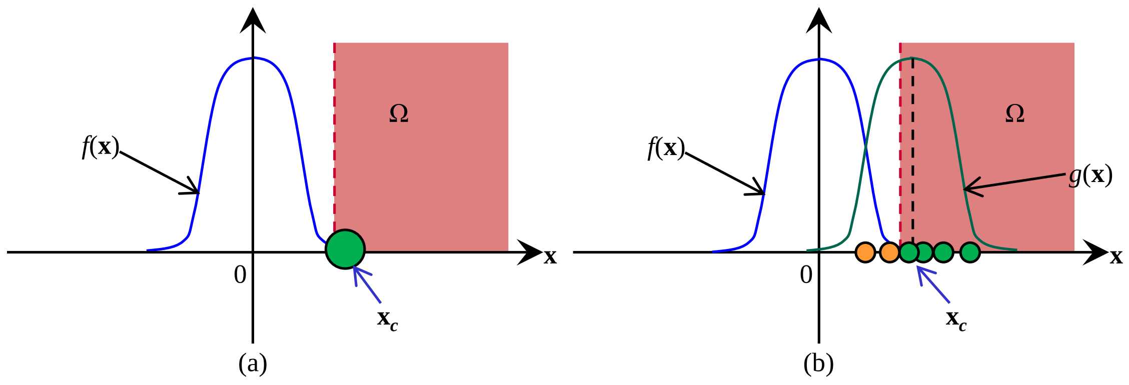

Figure 2-4. A simple 1-dimensional example is used to illustrate the key idea of MNIS described in Algorithm 2.3. Yellow points denote samples that do not belong to the failure region Ω, and green points denote samples that belong to the failure region Ω. (a) Find a failure sample $\mathbf{x}_c$ that has the minimum L_2-norm from the origin $\mathbf{x} = \mathbf{0}$. (b) Construct a shifted Normal distribution $g(\mathbf{x})$ with $\mathbf{x}_c$, and generate random samples from $g(\mathbf{x})$. Calculate failure rate with these random samples by using importance sampling equation (2.20).

Finding the minimum-norm failure sample is not trivial, especially if we have a large number of random variables in the vector $\mathbf{x}$. In other words, we cannot easily find the high-probability failure region in a high-dimensional variation space given that N_1 is small. Such issue is most pronounced, when the failure region of interest has a complicated (e.g., non-convex or even discontinuous) shape. To the best of our knowledge, there is no existing algorithm that can be generally applied to efficiently search a high-dimensional variation space. In a word, MNIS suffers from the dimensionality curse.

2.3.2.3 Gibbs Sampling Based Importance Sampling

Gibbs sampling based importance sampling (Gibbs) approach [45] also considers a Normal distribution as the distorted distribution. But unlike MIS and MNIS that only the mean of the distorted Normal distribution is estimated, both the mean and variance of the distorted Normal distribution are estimated by Gibbs. To this end, Gibbs applies a two-stage flow as follows [45]

Algorithm 2.4 Gibbs Sampling Based Importance Sampling

1. Generate a number of Gibbs samples from $g^{OPT}(\mathbf{x})$ in (2.22) by Gibbs sampling technique [72].
2. Compute the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$ of these Gibbs samples, and construct a Normal distribution $g(\mathbf{x})$

$$g\left(\mathbf{x}\right)=\frac{1}{\left(\sqrt{2\pi}\right)^{M}\cdot\left|\Sigma\right|^{0.5}}\cdot\exp\left[\frac{-\left(\mathbf{x}-\boldsymbol{\mu}\right)^{T}\cdot\Sigma^{-1}\cdot\left(\mathbf{x}-\boldsymbol{\mu}\right)}{2}\right]. \tag{2.26}$$

Generate N_2 random samples from $g(\mathbf{x})$ and estimate failure rate by (2.20).

To clearly understand Gibbs, let us consider a simple 1-dimensional example. Note that Gibbs assumes that the vector **x** in (2.13) follows a multivariate Normal distribution defined in (2.23). For this simple 1-dimensional example, $f(\mathbf{x})$ is plotted in Figure 2-5 (a). According to (2.22), the optimal distorted distribution $g^{OPT}(\mathbf{x})$ is plotted in Figure 2-5 (b). Since the failure region Ω is unknown in advance, we do not exactly know $g^{OPT}(\mathbf{x})$. Hence, we cannot directly draw random samples from $g^{OPT}(\mathbf{x})$. Gibbs applies Gibbs sampling technique [72] to generate a number of Gibbs samples by sampling a sequence of 1-dimensional conditional PDFs. Theoretically these Gibbs samples eventually follow the desired distribution $g^{OPT}(\mathbf{x})$. Details about how to generate Gibbs samples are omitted here, but can be found in the literature [45]. Based on these Gibbs samples, we compute their mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$. Next, we approximate $g^{OPT}(\mathbf{x})$ by a Normal distribution $g(\mathbf{x})$ in (2.26) as shown in Figure 2-5 (c), and perform importance sampling with $g(\mathbf{x})$ as shown in Figure 2-5 (d).

Note that Gibbs varies only one variable between two adjacent Gibbs samples. Hence, a large number of Gibbs samples are needed in a high-dimensional variation space. To create each Gibbs sample, Gibbs needs to construct a 1-dimensional conditional PDF by running binary search with a number of (e.g., 6~7) transistor-level simulations. Alternatively speaking, Gibbs requires 6~7 simulations to generate each Gibbs sample. Hence, we need to run a large number of transistor-level simulations to create enough Gibbs samples at Stage 1 of Algorithm 2.4 if we have many random variables in the vector **x**. From this point of view, Gibbs cannot be efficiently applied in a high-dimensional variation space.

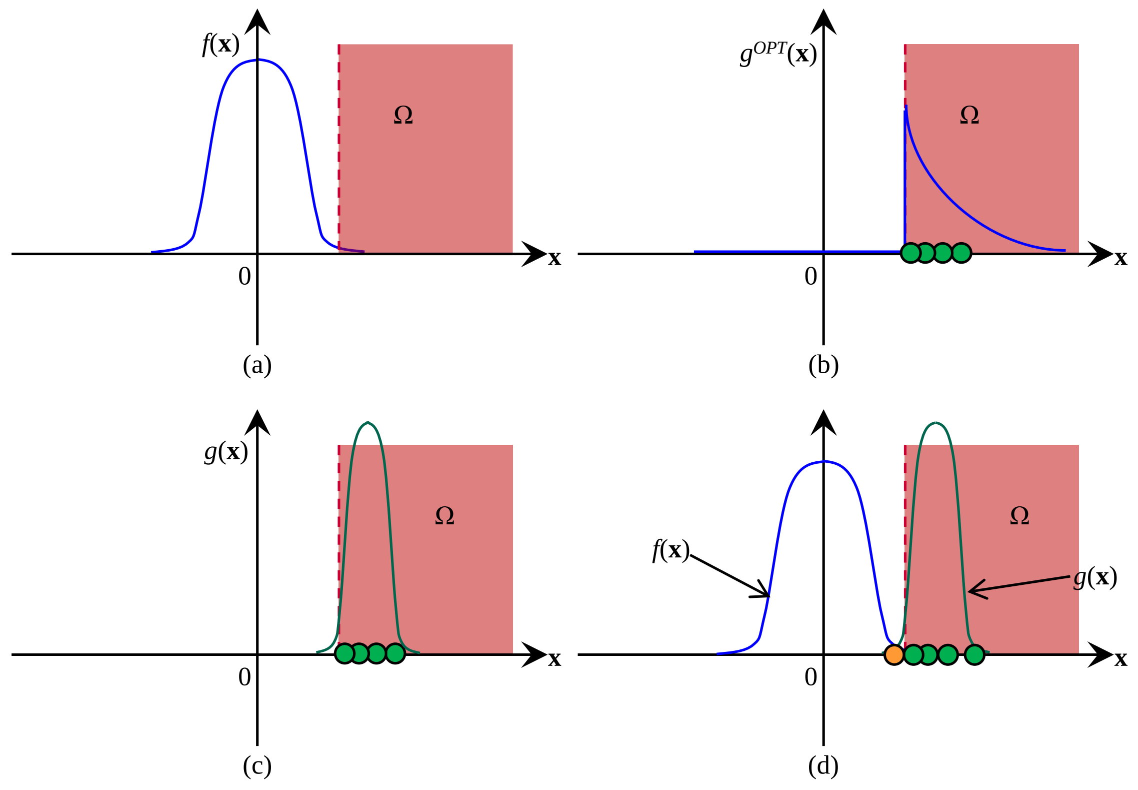

Figure 2-5. A simple 1-dimensional example is used to illustrate the key idea of Gibbs described in Algorithm 2.4. Yellow points denote samples that do not belong to the failure region Ω, and green points denote samples that belong to the failure region Ω. (a) $f(\mathbf{x})$ follows a standard Normal distribution. (b) Generate a number of Gibbs samples from the optimal distorted distribution $g^{OPT}(\mathbf{x})$. (c) Construct a Normal distribution $g(\mathbf{x})$ based on these Gibbs samples. (d) Generate random samples from $g(\mathbf{x})$, and calculate failure rate with these random samples by using importance sampling equation (2.20).

2.3.2.4 Sequential Monte Carlo

Sequential Monte Carlo (SMC) approach has been widely used in many areas [46]-[52], and was adopted to estimate rare failure rates in the CAD community by [39]. The key idea of SMC is to explore the failure region by a set of particles (i.e., samples), and estimate failure rates bases on these particles by using importance sampling technique. The simplified SMC flow is described as follows [39]

Algorithm 2.5 Sequential Monte Carlo

1. Start with a set of (say, N) particles $\{\mathbf{x}_1^{(n)}; n = 1, 2, \cdots, N\}$. Set $i = 1$.
2. Compute the weight $w[\mathbf{x}_i^{(n)}]$ for each particle $\mathbf{x}_i^{(n)}$ where $n \in 1, 2, \cdots, N$

$$w\left[\mathbf{x}_i^{(n)}\right] = f\left[\mathbf{x}_i^{(n)}\right] \cdot I\left[\mathbf{x}_i^{(n)}\right] \qquad n = 1, 2, \cdots, N \tag{2.27}$$

   where $f(\mathbf{x})$ denotes the PDF for $\mathbf{x}$ and $I(\mathbf{x})$ is defined in (2.16).
3. Resample $\{\mathbf{x}_i^{(n)}; n = 1, 2, \cdots, N\}$ based on $\{w[\mathbf{x}_i^{(n)}]; n = 1, 2, \cdots, N\}$. If a particle has a larger weight, it will have more repetitions after resampling. Denote $\{\mathbf{z}_i^{(n)}; n = 1, 2, \cdots, N\}$ as resampled particles.
4. Generate a new particle $\mathbf{x}_{i+1}^{(n)}$ from $\mathbf{z}_i^{(n)}$ by sampling a multivariate Normal distribution

$$\mathbf{x}_{i+1}^{(n)} \sim N\left[\mathbf{z}_i^{(n)}, \Sigma\right] \qquad n = 1, 2, \cdots, N \tag{2.28}$$

   where the covariance matrix Σ of the Normal distribution is pre-defined by the user.
5. Compute $g[\mathbf{x}_{i+1}^{(n)}]$ by

$$g\left[\mathbf{x}_{i+1}^{(n)}\right] = \frac{1}{N}\sum_{k=1}^{N} N\left[\mathbf{x}_{i+1}^{(n)} \middle| \mathbf{z}_i^{(k)}, \Sigma\right] \qquad n = 1, 2, \cdots, N \ . \tag{2.29}$$

   Estimate failure rate by importance sampling equation

$$P_i^{SMC} = \frac{1}{N} \cdot \sum_{n=1}^{N} \frac{f\left[\mathbf{x}_{i+1}^{(n)}\right] \cdot I\left[\mathbf{x}_{i+1}^{(n)}\right]}{g\left[\mathbf{x}_{i+1}^{(n)}\right]} \ . \tag{2.30}$$

   If the variation of $\{P_j^{SMC}; j = 1, 2, \cdots, i\}$ is small, the algorithm is complete. Otherwise, set $i = i + 1$, and go to Step 2.

SMC has successfully estimated failure rates for circuits which have a small number of random variables [39]. However, SMC described in Algorithm 2.5 suffers from the dimensionality curse [48]. Once we have a large number of random variables, the weights between different particles would differ a lot, resulting in particle collapse after resampling at Step 3. In other words, we end up with a single or very few particles after resampling and, hence, cannot efficiently explore the variation space.

The fundamental reason causing particle collapse is that particles are sampled from a high-dimensional distribution as shown in (2.28). To address the particle collapse issue, we can vary only one random variable when creating a new particle. However, since we do not know which random variable is more important for failure rate estimation in advance, we need to iterate through all the random variables. For each random variable, assume that we generate N particles. In total, we need to generate $N \times M$ particles during each iteration. Hence, this idea does not scale well once we have a large number of random variables.

2.3.3 Statistical Blockade

Statistical blockade (SB) is another approach to estimating rare failure events in the CAD community [34]. SB is a model-based approach, and its simplified algorithm flow is described as follows:

Algorithm 2.6 Statistical Blockade

```
1.  Generate a number of random samples $\{\mathbf{x}_n; n = 1, 2, \cdots, N_1\}$, and perform transistor-level
    simulations to evaluate their performance values. Denote performance values as $\{y(\mathbf{x}_n); n = 1, 2, \cdots, N_1\}$.
    Here, $y(\mathbf{x}_n)$ denotes the performance value for $\mathbf{x}_n$.
2.  Build a classifier based on the data set $\{\mathbf{x}_n, y(\mathbf{x}_n); n = 1, 2, \cdots, N_1\}$. The classifier is used to tell
    whether a sample is likely to fall into the failure region.
3.  Generate a large number of random samples from the natural PDF of $\mathbf{x}$ (i.e., $f(\mathbf{x})$). Evaluate these
    samples by the classifier learned at Step 2 of Algorithm 2.6. For samples that are identified as failure
    samples or very close to the failure boundary based on the classifier, SB further performs transistor-
    level simulations to evaluate these samples. Denote these samples as $\{\mathbf{x}_n^{MC}; n = 1, 2, \cdots, N_2\}$, and
    their performance values evaluated by transistor-level simulations as $\{y(\mathbf{x}_n^{MC}); n = 1, 2, \cdots, N_2\}$.
4.  Construct the tail distribution of our performance of interest with $\{y(\mathbf{x}_n^{MC}); n = 1, 2, \cdots, N_2\}$.
    Estimate the rare failure rate from the constructed tail distribution.
```

To clearly understand SB described in Algorithm 2.6, let us consider a simple 2-dimensional example. SB starts with a number of random samples and evaluates their performance values by running transistor-level simulations, as shown in Figure 2-6 (a). Here, yellow points denote samples that do not belong to the failure region Ω, and green points denote samples that belong to the failure region Ω.

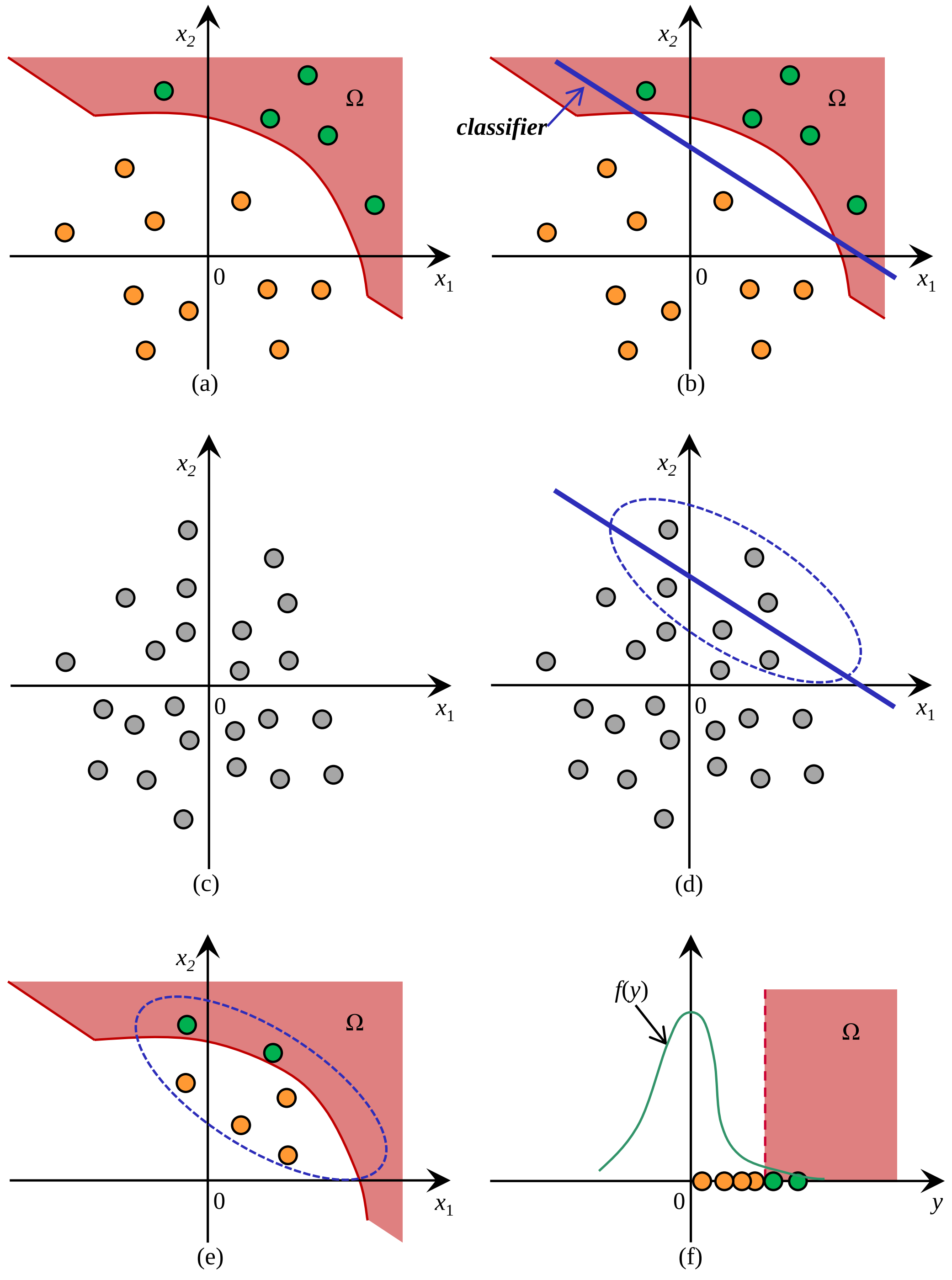

Figure 2-6. A simple 2-dimensional example is used to illustrate the key idea of SB described in Algorithm 2.6. Yellow points denote samples that do not belong to the failure region Ω, and green points denote samples that belong to the failure region Ω. Grey points denote random samples that are drawn from $f(\mathbf{x})$, and their performance values are unknown. (a) Generate a number of random samples and run transistor-

level simulations to evaluate their performance values. (b) Build a classifier based on random samples created in (a). (c) Generate a large number of random samples from the natural PDF $f(\mathbf{x})$. (d) For all the grey samples, identify samples that fall into the failure region Ω or are close to the failure boundary based on the classifier. These identified grey samples are circled by a dashed blue ellipsoid in this simple 2-dimensional example. (e) Evaluate these identified grey samples. (f) Learn the tail distribution of our performance of interest. Here, y is our performance of interest, and $f(y)$ denotes the PDF of y. Estimate rare failure rates based on the constructed tail distribution.

Based on these random samples, SB builds a classifier. In Figure 2-6 (b), we use the blue line to denote this classifier. Next, SB draws a large number of random samples from the original natural PDF $f(\mathbf{x})$. Here, we use grey points to denote these random samples drawn from $f(\mathbf{x})$, as shown in Figure 2-6 (c). Note that these grey points are not evaluated yet and, hence, their performance values are unknown. For all these grey points, identify samples that fall into the failure region Ω or are close to the failure boundary according to the classifier. In Figure 2-6 (d), these identified samples are circled by a dashed blue ellipsoid. Note that evaluating each grey point by the classifier is much faster than evaluating the same grey point by a transistor-level simulation. Therefore, evaluating all the grey points by the classifier does not consume a lot of time, and we can afford it. For all the identified grey points, we further run transistor-level simulations to evaluate their performance values, as shown in Figure 2-6 (e). If the number of identified grey points is small, simulation cost for evaluating these identified grey points is not large. Finally, SB learns the tail distribution of $f(y)$ based on those identified grey points and their simulation results, as shown in Figure 2-6 (f). Here, y is our performance of interest, and $f(y)$ denotes the distribution of y. Once the tail distribution of $f(y)$ is learned, SB calculates the rare failure rate by integrating the tail distribution.

SB has been successfully applied to estimate rare failure rates in a low-dimensional space. However, if we have a large number of random variables, SB can be pretty inefficient. The efficiency achieved by SB strongly depends on the accuracy of the classifier. As we know, constructing an accurate classifier in a high-dimensional space is generally challenging, especially when we cannot afford a large number of training samples (i.e., N_1 cannot be too large). If the classifier is not sufficiently accurate (e.g., the classifier boundary is far away from the real failure boundary), a lot of grey points could be identified at Step 3 of Algorithm 2.6, making SB extremely inefficient.

2.3.4 Integration-Based Approaches

The key idea of integration-based approaches [28]-[29], [38], [42] is to detect the failure boundary in the variation space, and then calculate the failure rate by integrating the PDF $f(\mathbf{x})$ over the failure region. The failure boundary is typically described by a number of random samples. The efficiency of these deterministic approaches is mainly determined by two things: (1) the shape of the failure boundary, and (2) the dimensionality of the variation space. If the failure boundary is simple to describe (e.g., linear boundary) or the dimensionality is low, we can achieve satisfactory accuracy and efficiency by applying these integration-based approaches. However, if the shape of the failure boundary is complicated and the dimensionality of the variation space is high, we need a large number of random samples to accurately describe the failure boundary. Note that each random sample is obtained by applying some search algorithms (e.g., binary search). Hence, obtaining each sample requires a few transistor-level simulations. As a result, the total number of simulations required by these integration-based approaches could quickly become intractable in a high-dimensional space especially if the failure region is complicated.

2.3.5 Limitations of Traditional Approaches

As discussed in Section 2.3.1-2.3.4, traditional approaches have their own application scenarios and also limitations. Selecting an appropriate estimation approach is nontrivial, and is determined by many factors:

1) The number of simulations that we can afford: this is determined by the number of simulator licenses that we have, our computing resources, simulation cost/time per simulation, project timeline, etc. The number of simulator licenses and computing resources determine how many simulations that we can perform simultaneously. Simulation time per simulation and project timeline determine how many simulations that we can run per machine. If the number of simulations required by the candidate approach is more than the maximum number of simulations that we can afford, we should not choose this candidate.

2) The number of simulations that are required: this is determined by the real failure rate that we are

interested in, the required estimation accuracy, the selected approach, circuit dimensionality, etc. If we select brute-force Monte Carlo approach and the real failure rate is P_f, the relative error δ can be written as

$$\delta = \frac{\sqrt{v_f^{MC}}}{P_f} \approx \frac{1}{P_f} \cdot \sqrt{\frac{P_f^{MC} \cdot \left(1 - P_f^{MC}\right)}{N}} \approx \sqrt{\frac{1}{N \cdot P_f}} \tag{2.31}$$

where P_f^{MC} and v_f^{MC} are defined in (2.17) and (2.18) respectively. From (2.31), we can observe that to achieve the required relative error δ, the required number of simulations N is proportional to $1/P_f$. If we choose statistical blockade, the required number of simulations is mainly determined by the number of training samples used for building the classifier and the number of identified random samples used for building the tail distribution, as discussed in Section 2.3.3. Both the number of training samples and the number of identified random samples are affected by dimensionality.

3) Dimensionality: this is determined by the circuit block that we are interested in and process design kit. If the circuit has only a few transistors and only a small number of local mismatch random variables are used to model local mismatches, the dimensionality of the variation space is low. Otherwise, we need to handle a high-dimensional variation space.

Considering the aforementioned factors, typical application scenarios for traditional approaches are described as follows:

- **Brute-force Monte Carlo**: (1) the target failure rate is large, and (2) the target failure rate is small and we can afford a large number of simulations. The second scenario typically occurs at the design sign-off stage.
- **Importance Sampling**: the target failure rate is small, the dimensionality is low and we want to spend only a small number of (e.g., a few thousand) simulations for failure rate estimation.
- **Statistical Blockade**: the target failure rate is small, the dimensionality is low and we want to spend only a small number of (e.g., a few thousand) simulations for failure rate estimation.
- **Integration-based Approaches**: the target failure rate is small, the dimensionality is low and we want to spend only a small number of (e.g., a few thousand) simulations for failure rate estimation.

As discussed in Section 2.2, we need efficient approaches to estimate rare failure events in a high-dimensional variation space. Unfortunately, such application scenario cannot be handled by most of these traditional approaches [24]-[45]. Therefore, there is a strong need of developing a new CAD tool to accurately capture the rare failure events in a high-dimensional variation space with low computational cost. To this end, we propose three novel approaches in this thesis. In the next three chapters, we will describe our proposed approaches in detail.

Chapter 3

Subset Simulation for Continuous Performance Metrics

To accurately capture the rare failure events in a high-dimensional variation space, a novel *subset simulation* (SUS) technique is proposed in this chapter. The key idea of SUS, borrowed from other communities [53]-[59], is to express the rare failure probability as the product of several large conditional probabilities by introducing a number of intermediate failure events. The original problem of rare failure probability estimation is then cast to an equivalent problem of estimating a sequence of conditional probabilities. Since these conditional probabilities are large, they are relatively easier to estimate than the original rare failure rate.

When implementing the proposed SUS method, it is difficult, if not impossible, to directly draw random samples from the conditional PDFs and estimate the conditional probabilities, since these conditional PDFs are unknown in advance. To address this issue, a modified Metropolis (MM) algorithm is adopted from the literature [54] to generate random samples by constructing a number of Markov chains. The conditional probabilities of interest are then estimated from these random samples. Unlike most traditional techniques [24]-[45] that suffer from the dimensionality issue, SUS can be efficiently applied to high-dimensional problems, which will be demonstrated by the experimental results in Section 3.5.

It is important to emphasize that though the key idea of SUS was first proposed in other communities [53]-[59], how to estimate the confidence interval of SUS remains an open question. Due to this reason, the SUS method has not been successfully applied to many practical applications in the literature. In this chapter, we propose a statistical methodology to accurately estimate the confidence interval of SUS based on the theory of Markov chain Monte Carlo simulation, thereby making our proposed SUS method of practical utility.

Without loss of generality, we assume that the performance function $y(\mathbf{x})$ is continuous and F stands for the failure region in the performance space (i.e., the subset of the performance space where the performance of interest y does not meet the specification)

$$F = \{y(\mathbf{x}) | \mathbf{x} \in \Omega\}. \tag{3.1}$$

Given (3.1), the rare failure rate P_f defined in (2.14) can be re-written as

$$P_f = \Pr\left[y(\mathbf{x}) \in F\right] \tag{3.2}$$

where Pr(•) denotes the probability that an event occurs. Define K intermediate failure events $\{F_k;\ k = 1, 2, \cdots, K\}$ as the subsets of the performance space

$$F = F_K \subset F_{K-1} \subset \cdots \subset F_2 \subset F_1. \tag{3.3}$$

Based on (3.3), we can express P_f in (3.2) as

$$P_f = \Pr\left[y(\mathbf{x}) \in F\right] = \Pr\left[y(\mathbf{x}) \in F_K\right] = \Pr\left[y(\mathbf{x}) \in F_K, y(\mathbf{x}) \in F_{K-1}\right]. \tag{3.4}$$

In (3.4), if $y(\mathbf{x})$ belongs to F_K, it will undoubtedly belong to F_{K-1} because F_K is a subset of F_{K-1} as shown in (3.3). Eq. (3.4) can be re-written as [71]

$$P_f = \Pr\left[y(\mathbf{x}) \in F_K \middle| y(\mathbf{x}) \in F_{K-1}\right] \cdot \Pr\left[y(\mathbf{x}) \in F_{K-1}\right]. \tag{3.5}$$

Here, the conditional probability $\Pr[y(\mathbf{x}) \in F_K \mid y(\mathbf{x}) \in F_{K-1}]$ represents the probability of $y(\mathbf{x}) \in F_K$ given $y(\mathbf{x}) \in F_{K-1}$. Similarly, we can express $\Pr[y(\mathbf{x}) \in F_{K-1}]$ as

$$\Pr\left[y(\mathbf{x}) \in F_{K-1}\right] = \Pr\left[y(\mathbf{x}) \in F_{K-1} \middle| y(\mathbf{x}) \in F_{K-2}\right] \cdot \Pr\left[y(\mathbf{x}) \in F_{K-2}\right]. \tag{3.6}$$

From (3.3), (3.5) and (3.6), we can easily derive

$$P_f = \Pr\left[y(\mathbf{x}) \in F_1\right] \cdot \prod_{k=2}^{K} \Pr\left[y(\mathbf{x}) \in F_k \middle| y(\mathbf{x}) \in F_{k-1}\right] = \prod_{k=1}^{K} P_k \tag{3.7}$$

where

$$P_1 = \Pr\left[y(\mathbf{x}) \in F_1\right] \tag{3.8}$$

$$P_k = \Pr\left[y(\mathbf{x}) \in F_k \middle| y(\mathbf{x}) \in F_{k-1}\right] \quad (k = 2, 3, \cdots, K). \tag{3.9}$$

If the failure events $\{F_k;\ k = 1, 2, \cdots, K\}$ are properly chosen, all the probabilities $\{P_k;\ k = 1, 2, \cdots, K\}$ are large and can be efficiently estimated. Once $\{P_k;\ k = 1, 2, \cdots, K\}$ are known, the rare failure probability P_f can be easily calculated by (3.7). In what follows, we will first use a simple 2-dimensional example to intuitively illustrate the basic flow of SUS in Section 3.1, and then further generalize SUS to high dimension in Section 3.2.

3.1 A Simple 2-D Example

Figure 3-1 shows a simple 2-dimensional example with two random variables $\mathbf{x} = [x_1\ x_2]$ to model the device-level process variations. In Figure 3-1, the failure regions Ω_1 and Ω_2 denote the subsets of the variation space where the performance of interest $y(\mathbf{x})$ belongs to F_1 and F_2 respectively, i.e., $\Omega_1 = \{\mathbf{x} \mid y(\mathbf{x}) \in F_1\}$ and $\Omega_2 = \{\mathbf{x} \mid y(\mathbf{x}) \in F_2\}$. Note that Ω_1 and Ω_2 are depicted for illustration purposes in this example. In practice, we do not need to explicitly know Ω_1 and Ω_2. Instead, we can run a transistor-level simulation to determine whether a sample $\mathbf{x}$ belongs to Ω_1 and/or Ω_2.

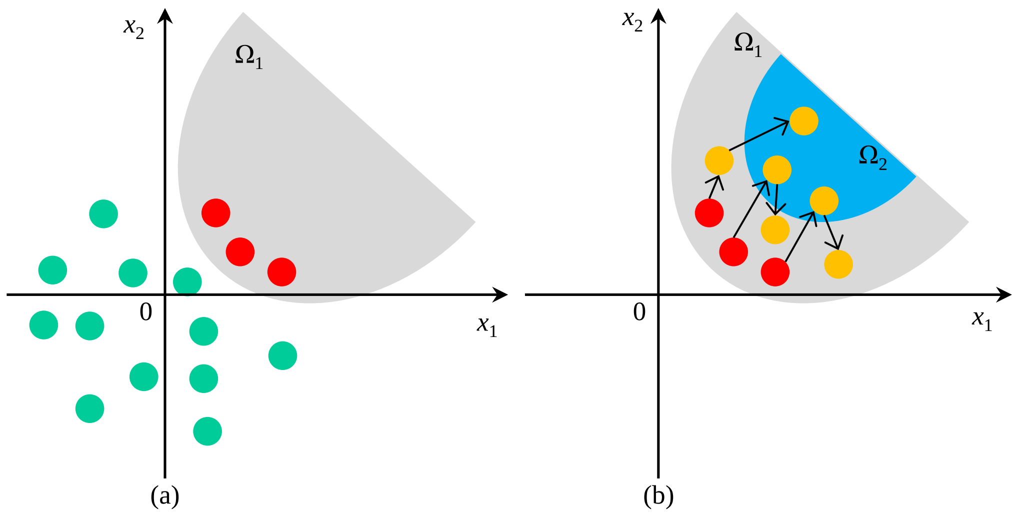

Figure 3-1. A 2-dimensional example is used to illustrate the procedure of probability estimation via multiple phases by using SUS: (a) generating MC samples and estimating P_1 in the 1st phase, and (b) generating MCMC samples and estimating P_2 in the 2nd phase.

Our objective is to estimate the probabilities $\{P_k; k = 1, 2, \cdots, K\}$ via multiple phases. Starting from the 1st phase, we simply draw L_1 independent random samples $\{\mathbf{x}^{(1,l)}; l = 1, 2, \cdots, L_1\}$ from the PDF $f(\mathbf{x})$ to estimate P_1. Here, the superscript "1" of the symbol $\mathbf{x}^{(1,l)}$ refers to the 1st phase. Among these L_1 samples, we identify a subset of samples $\{\mathbf{x}_F^{(1,t)}; t = 1, 2, \cdots, T_1\}$ that fall into Ω_1, where T_1 denotes the total number of the samples in this subset. As shown in Figure 3-1 (a), the red points represent the samples that belong to

Ω_1 and the green points represent the samples that are out of Ω_1. In this case, P_1 can be estimated as

$$P_1^{SUS} = \frac{1}{L_1} \cdot \sum_{l=1}^{L_1} I_{F_1}\left[y\left(\mathbf{x}^{(1,l)}\right)\right] = \frac{T_1}{L_1} \tag{3.10}$$

where P_1^{SUS} denotes the estimated value of P_1, and $I_{F1}[y(\mathbf{x})]$ represents the indicator function

$$I_{F_1}\left[y(\mathbf{x})\right] = \begin{cases} 1 & y(\mathbf{x}) \in F_1 \\ 0 & y(\mathbf{x}) \notin F_1 \end{cases}. \tag{3.11}$$

If P_1 is large, it can be accurately estimated by the aforementioned brute-force Monte Carlo method with a small number of random samples (e.g., L_1 is around 10^2~10^3).

Next, in the 2nd phase, we need to estimate the conditional probability $P_2 = \Pr[y(\mathbf{x}) \in F_2 \mid y(\mathbf{x}) \in F_1]$ which can be re-written as $\Pr(\mathbf{x} \in \Omega_2 \mid \mathbf{x} \in \Omega_1)$. Towards this goal, one simple idea is to directly draw random samples from the conditional PDF $f(\mathbf{x} \mid \mathbf{x} \in \Omega_1)$ and then compute the mean of the indicator function $I_{F2}[y(\mathbf{x})]$

$$I_{F_2}\left[y(\mathbf{x})\right] = \begin{cases} 1 & y(\mathbf{x}) \in F_2 \\ 0 & y(\mathbf{x}) \notin F_2 \end{cases}. \tag{3.12}$$

This approach, however, is practically infeasible since $f(\mathbf{x} \mid \mathbf{x} \in \Omega_1)$ is unknown in advance. To address this challenge, we apply a modified Metropolis (MM) algorithm [54] to generate a set of random samples that follow the conditional PDF $f(\mathbf{x} \mid \mathbf{x} \in \Omega_1)$.

MM is a Markov chain Monte Carlo (MCMC) technique [72]. Starting from each of the samples $\{\mathbf{x}_F^{(1,\,t)};\ t = 1, 2, \cdots, T_1\}$ that fall into Ω_1 in the 1st phase, MM generates a sequence of samples that form a Markov chain. In other words, there are T_1 independently generated Markov chains in total and $\mathbf{x}_F^{(1,\,t)}$ is the 1st sample of the t-th Markov chain. To clearly explain the MM algorithm, we define the symbol $\mathbf{x}^{(2,\,t,\,1)} = \mathbf{x}_F^{(1,\,t)}$, where $t \in \{1, 2, \cdots, T_1\}$. The superscripts "2" and "1" of $\mathbf{x}^{(2,\,t,\,1)}$ refer to the 2nd phase and the 1st sample of the Markov chain respectively.

For our 2-dimensional example, we start from $\mathbf{x}^{(2,\,1,\,1)} = [x_1^{(2,\,1,\,1)}\ x_2^{(2,\,1,\,1)}]$ to form the 1st Markov chain. To generate the 2nd sample $\mathbf{x}^{(2,\,1,\,2)}$ from $\mathbf{x}^{(2,\,1,\,1)}$, we first randomly sample a new value x_1^{NEW} from a 1-dimensional transition PDF $q_1[x_1^{NEW} \mid x_1^{(2,\,1,\,1)}]$ that must satisfy the following condition [54]

$$q_1\left[x_1^{NEW} \middle| x_1^{(2,1,1)}\right] = q_1\left[x_1^{(2,1,1)} \middle| x_1^{NEW}\right]. \tag{3.13}$$

There are many possible ways to define $q_1[x_1^{NEW} \mid x_1^{(2,\,1,\,1)}]$ in (3.13). For example, a 1-dimensional Normal

PDF can be used

$$q_1\left[x_1^{NEW}\middle|x_1^{(2,1,1)}\right]=\frac{1}{\sqrt{2\pi}\cdot\sigma_1}\cdot\exp\left\{-\frac{\left[x_1^{NEW}-x_1^{(2,1,1)}\right]^2}{2\cdot\sigma_1^2}\right\} \tag{3.14}$$

where $x_1^{(2,1,1)}$ and σ_1 are the mean and standard deviation of the distribution respectively. Here, σ_1 is a parameter that should be empirically chosen by following the heuristics in Section 3.2.

Next, we compute the ratio

$$r=\frac{f_1\left[x_1^{NEW}\right]}{f_1\left[x_1^{(2,1,1)}\right]} \tag{3.15}$$

where $f_1(x_1)$ is the original PDF of the random variable x_1. Since all the random variables defined in (2.13) are independent, $f(\mathbf{x})$ can be written as

$$f(\mathbf{x})=\prod_{m=1}^{M}f_m(x_m) \tag{3.16}$$

where $f_m(x_m)$ denotes the PDF of x_m. A random sample u is then drawn from a 1-dimensional uniform distribution with the following PDF

$$U(u)=\begin{cases}1 & 0\le u\le 1\\ 0 & \text{Otherwise}\end{cases} \tag{3.17}$$

and the value of $x_1^{(2,1,2)}$ is set as

$$x_1^{(2,1,2)}=\begin{cases}x_1^{NEW} & u\le\min(1,r)\\ x_1^{(2,1,1)} & u>\min(1,r)\end{cases}. \tag{3.18}$$

A similar procedure is applied to generate $x_2^{(2,1,2)}$. Once $x_1^{(2,1,2)}$ and $x_2^{(2,1,2)}$ are determined, we form a candidate $\mathbf{x}^{NEW}=[x_1^{(2,1,2)}\ x_2^{(2,1,2)}]$ and use it to create the sample $\mathbf{x}^{(2,1,2)}$

$$\mathbf{x}^{(2,1,2)}=\begin{cases}\mathbf{x}^{NEW} & \mathbf{x}^{NEW}\in\Omega_1\\ \mathbf{x}^{(2,1,1)} & \mathbf{x}^{NEW}\notin\Omega_1\end{cases}. \tag{3.19}$$

By repeating the aforementioned steps, we can create other samples to complete the Markov chain $\{\mathbf{x}^{(2,1,l)};\ l=1,2,\cdots,L_2\}$, where L_2 denotes the length of the Markov chain in the 2nd phase. In addition, all other Markov chains can be similarly formed. Since there are T_1 Markov chains and each Markov chain contains L_2 samples, the total number of the MCMC samples is $T_1\cdot L_2$ for the 2nd phase. Figure 3-1 (b) shows the sampling results for our 2-dimensional example. In Figure 3-1 (b), the red points represent the initial samples $\{\mathbf{x}^{(2,t,1)}; t = 1, 2, \cdots, T_1\}$ of the Markov chains and they are obtained from the 1st phase. The yellow points represent the MCMC samples created via the MM algorithm in the 2nd phase.

It has been proved in [54] that if the initial sample $\mathbf{x}^{(2,t,1)}$ follows the distribution $f(\mathbf{x} \mid \mathbf{x} \in \Omega_1)$, all the samples $\{\mathbf{x}^{(2,t,l)}; l = 1, 2, \cdots, L_2\}$ in the Markov chain follow $f(\mathbf{x} \mid \mathbf{x} \in \Omega_1)$. In our 2-dimensional example, since the initial samples $\{\mathbf{x}^{(2,t,1)}; t = 1, 2, \cdots, T_1\}$ are randomly drawn from $f(\mathbf{x})$ and belong to Ω_1, they follow the distribution $f(\mathbf{x} \mid \mathbf{x} \in \Omega_1)$. Hence, all the MCMC samples $\{\mathbf{x}^{(2,t,l)}; t = 1, 2, \cdots, T_1, l = 1, 2, \cdots, L_2\}$ in Figure 3-1 (b) follow $f(\mathbf{x} \mid \mathbf{x} \in \Omega_1)$. In other words, we have successfully generated a number of random samples that follow our desired distribution for the 2nd phase.

Among all the MCMC samples $\{\mathbf{x}^{(2,t,l)}; t = 1, 2, \cdots, T_1, l = 1, 2, \cdots, L_2\}$, we further identify a subset of samples $\{\mathbf{x}_F^{(2,t)}; t = 1, 2, \cdots, T_2\}$ that fall into Ω_2, where T_2 denotes the total number of the samples in this subset. The conditional probability P_2 can be estimated as

$$P_2^{SUS} = \frac{1}{T_1 \cdot L_2} \cdot \sum_{t=1}^{T_1} \sum_{l=1}^{L_2} I_{F_2}\left[y\left(\mathbf{x}^{(2,t,l)} \right) \right] = \frac{T_2}{T_1 \cdot L_2} \tag{3.20}$$

where P_2^{SUS} denotes the estimated value of P_2, and $I_{F2}[y(\mathbf{x})]$ is the indicator function defined in (3.12).

By following the aforementioned idea, we can estimate all the probabilities $\{P_k; k = 1, 2, \cdots, K\}$. Namely, for the k-th phase where $k > 2$, we need to estimate the conditional probability $P_k = \Pr[y(\mathbf{x}) \in F_k \mid y(\mathbf{x}) \in F_{k-1}]$ by generating MCMC samples via the MM algorithm. Once the values of $\{P_k; k = 1, 2, \cdots, K\}$ are estimated, the rare failure rate P_f is calculated by

$$P_f^{SUS} = \prod_{k=1}^{K} P_k^{SUS} \tag{3.21}$$

where P_f^{SUS} represents the estimated value of P_f by using SUS.

3.2 High-Dimensional Case

If we have more than two random variables, estimating the probabilities $\{P_k; k = 1, 2, \cdots, K\}$ can be pursued in a similar way. Algorithm 3.1 summarizes the major steps of the proposed SUS method for high dimension. It consists of K phases. During the 1st phase, we randomly sample the PDF $f(\mathbf{x})$ to estimate P_1. Next, we apply MM (i.e., Step 7~15 in Algorithm 3.1) to estimate the conditional probability P_k during the k-th phase, where $k \in \{2, 3, \cdots, K\}$. When estimating P_k, we construct T_{k-1} Markov chains. Each Markov chain contains L_k MCMC samples $\{\mathbf{x}^{(k,\,t,\,l)};\ l = 1, 2, \cdots, L_k\}$ that are created by the MM algorithm. Hence, there are $T_{k-1}{\cdot}L_k$ samples in total, and the probability P_k is estimated by these samples. Finally, the rare failure rate P_f is estimated from $\{P_k;\ k = 1, 2, \cdots, K\}$ by using (3.21).

Algorithm 3.1 Subset Simulation (SUS)

```
1.  Start from a set of pre-defined failure events $\{F_k;\ k = 1, 2, \cdots, K\}$.
2.  Generate $L_1$ random samples $\{\mathbf{x}^{(1,\,l)};\ l = 1, 2, \cdots, L_1\}$ from $f(\mathbf{x})$.
3.  From the random samples $\{\mathbf{x}^{(1,\,l)};\ l = 1, 2, \cdots, L_1\}$, identify the samples for which $y[\mathbf{x}^{(1,\,l)}] \in F_1$. Label
    these samples as $\{\mathbf{x}_F^{(1,\,t)};\ t = 1, 2, \cdots, T_1\}$, where $T_1$ represents the total number of samples satisfying
    the condition. Calculate $P_1^{SUS}$ by (3.10).
4.  Initialize $k = 2$.
5.  Set $\mathbf{x}^{(k,\,t,\,1)} = \mathbf{x}_F^{(k-1,\,t)}$, where $t = 1, 2, \cdots, T_{k-1}$.
6.  For $t = 1, 2, \cdots, T_{k-1}$
7.      For $l = 2, 3, \cdots, L_k$
8.          For $m = 1, 2, \cdots, M$
9.              Generate a random value $x_m^{NEW}$ from the 1-D transition PDF $q_m[x_m^{NEW} \mid x_m^{(k,\,t,\,l-1)}]$. For
                instance, the 1-D transition PDF can be a Normal distribution
```

$$q_m\left[x_m^{NEW}\left|x_m^{(k,t,l-1)}\right.\right]=\frac{1}{\sqrt{2\pi}\cdot\sigma_m}\cdot\exp\left\{-\frac{\left[x_m^{NEW}-x_m^{(k,t,l-1)}\right]^2}{2\cdot\sigma_m^2}\right\}. \tag{3.22}$$

```
10.             Calculate the ratio
```

$$r=\frac{f_m\left[x_m^{NEW}\right]}{f_m\left[x_m^{(k,t,l-1)}\right]}. \tag{3.23}$$

```
11.             Draw a random value $u$ from the uniform distribution in (3.17) and set the value of $x_m^{(k,\,t,\,l)}$
                as
```

$$x_m^{(k,t,l)}=\begin{cases}x_m^{NEW} & u\le\min(1,r)\\ x_m^{(k,t,l-1)} & u>\min(1,r)\end{cases}. \tag{3.24}$$

```
12.         End For
13.         Form a candidate $\mathbf{x}^{NEW} = [x_1^{(k,\,t,\,l)}\ x_2^{(k,\,t,\,l)}\ \cdots\ x_M^{(k,\,t,\,l)}]$, and run a transistor-level simulation to
```

```
            evaluate $y(\mathbf{x}^{NEW})$.
14.         Set the value of $\mathbf{x}^{(k, t, l)}$ as
```

$$\mathbf{x}^{(k,t,l)} = \begin{cases} \mathbf{x}^{NEW} & y\left(\mathbf{x}^{NEW}\right) \in F_{k-1} \\ \mathbf{x}^{(k,t,l-1)} & y\left(\mathbf{x}^{NEW}\right) \notin F_{k-1} \end{cases}. \tag{3.25}$$

```
15.       End for
16.   End For
17.   From the $T_{k-1} \cdot L_k$ MCMC samples $\{\mathbf{x}^{(k, t, l)}; t = 1, 2, \cdots, T_{k-1}, l = 1, 2, \cdots, L_k\}$, identify the samples for
      which $y[\mathbf{x}^{(k, t, l)}] \in F_k$. Label these samples as $\{\mathbf{x}_F^{(k, t)}; t = 1, 2, \cdots, T_k\}$, where $T_k$ represents the total
      number of samples satisfying the condition.
18.   Calculate $P_k^{SUS}$ by
```

$$P_k^{SUS} = \frac{T_k}{T_{k-1} \cdot L_k}. \tag{3.26}$$

```
19.   If $k < K$, set $k = k + 1$ and go to Step 5. Otherwise, go to Step 20.
20.   Calculate $P_f^{SUS}$ by (3.21).
```

There are several important clarifications that should be made for Algorithm 3.1. First, sampling the transition PDF $q_m[x_m^{NEW} \mid x_m^{(k, t, l-1)}]$ in (3.22) at Step 9 or the uniform PDF $U(u)$ in (3.17) at Step 11 involves no transistor-level simulation and, hence, its computational cost is almost negligible. The computational cost of Algorithm 3.1 is dominated by the transistor-level simulation to evaluate $y(\mathbf{x}^{NEW})$ at Step 13.

Second, MM successively samples a set of 1-D transition PDFs $\{q_m[x_m^{NEW} \mid x_m^{(k, t, l-1)}]; m = 1, 2, \cdots, M\}$, instead of any high-dimensional joint PDF, to generate a new MCMC sample. For this reason, MM does not suffer from any dimensionality issue and can efficiently handle high-dimensional problems in practice. More detailed discussions about the efficiency of MM for high dimension can be found in [54].

Finally, the 1-D transition PDFs $\{q_m[x_m^{NEW} \mid x_m^{(k, t, l-1)}]; m = 1, 2, \cdots, M\}$ play an important role in sampling the failure region Ω_{k-1} at the k-th phase, where Ω_{k-1} denotes the subset of the variation space $\{\mathbf{x} \mid y(\mathbf{x}) \in F_{k-1}\}$. For illustration purposes, we consider the 1-D transition PDF in (3.22) as an example. For this Normal distribution, if its standard deviation σ_m is too large, it is likely that the new sample x_m^{NEW} is far away from the previous sample $x_m^{(k,\,t,\,l-1)}$. In other words, we attempt to “jump” over a long distance via the Markov chain. However, the new sample $\mathbf{x}^{NEW}$ may eventually fall out of the failure region Ω_{k-1} (i.e., $y(\mathbf{x}^{NEW}) \notin F_{k-1}$) and get rejected, as shown in (3.25).

On the other hand, if σ_m is too small, it is likely that the new sample x_m^{NEW} is extremely close to the previous sample $x_m^{(k,\,t,\,l-1)}$. In this case, it may require many MCMC samples to fully explore the failure region Ω_{k-1}. The aforementioned discussions imply an important fact that the value of σ_m must be appropriately chosen in order to make Algorithm 3.1 efficient. As a heuristic approach [54], we simply set σ_m equal to the standard deviation of the original PDF $f_m(x_m)$ shown in (3.16). Intuitively, if the standard deviation of $f_m(x_m)$ is large, the random variable x_m can vary over a large range. In this case, we want to set σ_m to a relatively large value so that the resulting Markov chain can quickly explore a large region of the variation space.

In the following sub-sections, we further discuss several important implementation issues for SUS, including (i) subset selection, and (ii) confidence interval estimation.

3.3 Subset Selection

In Algorithm 3.1, we assume that the failure events $\{F_k;\ k = 1, 2, \cdots, K\}$ are pre-defined. In practice, however, we have to carefully define $\{F_k;\ k = 1, 2, \cdots, K\}$ so that our proposed SUS method is computationally efficient. Otherwise, if $\{F_k;\ k = 1, 2, \cdots, K\}$ are not appropriately chosen, $\{P_k;\ k = 1, 2, \cdots, K\}$ can be extremely small and, hence, cannot be efficiently estimated by a small number of samples.

Without loss of generality, let us assume that we define K subsets $\{F_k;\ k = 1, 2, \cdots, K\}$ before running SUS, and generate N_k samples when estimating P_k where $k \in \{1, 2, \cdots, K\}$. Our objective is to minimize the estimation error given the pre-defined total number of samples N

$$\begin{aligned} &\min_{\substack{N_k, P_k \\ k=1,2,\cdots,K}} \quad \operatorname{var}\left(P_f^{SUS}\right) \\ &s.t. \quad \begin{cases} P_f = \prod_{k=1}^{K} P_k \\ N = \sum_{k=1}^{K} N_k \\ N_k > 0 \quad (k = 1, 2, \cdots, K) \\ 1 > P_k > 0 \quad (k = 1, 2, \cdots, K) \end{cases} \end{aligned} \tag{3.27}$$

where P_k is defined in (3.8) and (3.9), P_f^{SUS} is defined in (3.21), and var(P_f^{SUS}) denotes the variance of our proposed SUS estimator. As will be seen in Section 3.4, deriving the analytical expression for var(P_f^{SUS}) is extremely difficult, if not impossible, which makes the optimization problem in (3.27) hard to solve. To make the optimization problem in (3.27) solvable, we assume that

1) All the MCMC samples used for calculating P_k^{SUS} in (3.26) are statistically independent where $k \in \{2, 3, \cdots, K\}$.

2) $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$ are statistically independent.

Given these two assumptions, the optimization problem in (3.27) will be solvable, as will be seen later. Note that the solution that we get for the optimization problem in (3.27) is not the real optimal solution due to these two assumptions that we make. However, such solution can provide us with useful guidance on how to select subsets. In what follows, we will discuss how to solve the optimization problem in (3.27) given these two assumptions.

Since P_f^{SUS} is equal to the multiplication of $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$, we should carefully study the statistical property of $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$ in order to derive the variance for P_f^{SUS}. Given the first assumption that we make, the statistical distributions for $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$ can be written as [71]

$$P_k^{SUS} \sim N\left[P_k, \frac{P_k \cdot (1 - P_k)}{N_k}\right] \quad k = 1, 2, \cdots, K . \tag{3.28}$$

To further derive the distribution for P_f^{SUS} in (3.21) based on (3.28), we take logarithm on both sides of (3.21) because it is much easier to handle summation than multiplication

$$\log\left(P_f^{SUS}\right) = \sum_{k=1}^{K} \log\left(P_k^{SUS}\right). \tag{3.29}$$

To derive the distribution for $\{\log(P_k^{SUS}); k = 1, 2, \cdots, K\}$, we approximate the nonlinear function log(•) by the first-order Taylor expansion around the mean value P_k of the random variable P_k^{SUS}

$$\log\left(P_k^{SUS}\right) \approx \log\left(P_k\right) + \frac{P_k^{SUS} - P_k}{P_k}. \tag{3.30}$$

According to (3.28) and (3.30), log(P_k^{SUS}) follows a Normal distribution

$$\log\left(P_k^{SUS}\right) \sim N\left[\log\left(P_k\right), \frac{1-P_k}{N_k \cdot P_k}\right] \quad \left(k = 1, 2, \cdots, K\right). \tag{3.31}$$

Given the second assumption that {P_k^{SUS}; k = 1, 2, ⋯, K} are statistically independent, {log(P_k^{SUS}); k = 1, 2, ⋯, K} are also statistically independent. Since log(P_f^{SUS}) is the summation of {log(P_k^{SUS}); k = 1, 2, ⋯, K}, var[log(P_f^{SUS})] can be expressed as

$$\text{VAR}\left[\log\left(P_f^{SUS}\right)\right] = \text{VAR}\left[\sum_{k=1}^{K} \log\left(P_k^{SUS}\right)\right] = \sum_{k=1}^{K} \frac{1-P_k}{N_k \cdot P_k}. \tag{3.32}$$

Instead of minimizing var(P_f^{SUS}), we can minimize var[log(P_f^{SUS})]. Hence, the optimization problem in (3.27) becomes

$$\begin{aligned} &\min_{\substack{N_k, P_k \\ k=1,2,\cdots,K}} \quad obj = \sum_{k=1}^{K} \frac{1-P_k}{N_k \cdot P_k} \\ &s.t. \quad \begin{cases} P_f = \prod_{k=1}^{K} P_k \\ N = \sum_{k=1}^{K} N_k \\ N_k > 0 \quad \left(k = 1, 2, \cdots, K\right) \\ 1 > P_k > 0 \quad \left(k = 1, 2, \cdots, K\right) \end{cases} \end{aligned}. \tag{3.33}$$

The constrained optimization problem in (3.33) can be re-written as:

$$\begin{aligned} &\min_{\substack{N_k, P_k \\ k=1,2,\cdots,K-1}} \quad obj = \frac{1}{N_1} \cdot \left(\frac{1}{P_1} - 1\right) + \cdots + \frac{1}{N_{K-1}} \cdot \left(\frac{1}{P_{K-1}} - 1\right) + \frac{1}{N - N_1 - \cdots - N_{K-1}} \cdot \left(\frac{P_1 \cdots P_{K-1}}{P_f} - 1\right) \\ &s.t. \quad \begin{cases} N_k > 0 \quad \left(k = 1, 2, \cdots, K-1\right) \\ 1 > P_k > 0 \quad \left(k = 1, 2, \cdots, K-1\right) \end{cases} \end{aligned}. \tag{3.34}$$

Taking the first-order derivative of the objective function *obj* in (3.34) with respect to P_1 yields

$$\frac{\partial obj}{\partial P_1} = -\frac{1}{N_1 \cdot P_1^2} + \frac{1}{N - N_1 - \cdots - N_{K-1}} \cdot \frac{P_2 \cdots P_{K-1}}{P_f}. \tag{3.35}$$

Set the derivative in (3.35) to 0, and we have

$$\frac{\partial obj}{\partial P_1} = 0 \Rightarrow N_1 \cdot P_1 = N_K \cdot P_K. \tag{3.36}$$

Similarly, we can get

$$N_1 \cdot P_1 = N_2 \cdot P_2 = \cdots = N_K \cdot P_K = C \tag{3.37}$$

where C is introduced to simplify the notation. Based on (3.37), we have

$$N_k = \frac{C}{P_k} \quad k = 1, 2, \cdots, K . \tag{3.38}$$

Given (3.38) and the second constraint in (3.33), C can be expressed as

$$C = \frac{N}{\sum_{k=1}^{K} 1/P_k} . \tag{3.39}$$

Substituting (3.38) and (3.39) into (3.34) yields

$$\begin{aligned} &\min_{\substack{N_k, P_k \\ k=1,2,\cdots,K-1}} && obj = \frac{K}{N} \cdot \left(\frac{1}{P_1} + \cdots + \frac{1}{P_{K-1}} + \frac{P_1 \cdots P_{K-1}}{P_f} \right) - \left(\frac{1}{N_1} + \cdots + \frac{1}{N_{K-1}} + \frac{1}{N - N_1 - \cdots - N_{K-1}} \right) \\ &s.t. && \begin{cases} N_k > 0 & (k = 1, 2, \cdots, K-1) \\ 1 > P_k > 0 & (k = 1, 2, \cdots, K-1) \end{cases} \end{aligned} . \tag{3.40}$$

Take the first-order derivative of the objective function *obj* in (3.40) with respect to P_1, yielding

$$\frac{\partial obj}{\partial P_1} = \frac{K}{N} \left(-\frac{1}{P_1^2} + \frac{P_2 \cdots P_{K-1}}{P_f} \right) . \tag{3.41}$$

Set the derivative in (3.41) to 0, and we have

$$\frac{\partial obj}{\partial P_1} = 0 \Rightarrow P_1 = P_K . \tag{3.42}$$

Similarly, we can get

$$P_1 = P_2 = \cdots = P_K = P_f^{1/K} . \tag{3.43}$$

Based on (3.38) and (3.43), we have

$$N_1 = N_2 = \cdots = N_K = \frac{N}{K} . \tag{3.44}$$

Eq. (3.43) and (3.44) imply that we should select subsets in a way that all the conditional probabilities $\{P_k; k = 1, 2, \cdots, K\}$ are the same. In addition, we should assign the same number of samples when estimating $\{P_k; k = 1, 2, \cdots, K\}$.

However, we are still not clear about how to select subsets since we do not exactly know $\{P_k; k = 1, 2, \cdots, K\}$. To find the best value for $\{P_k; k = 1, 2, \cdots, K\}$, we substitute (3.43) and (3.44) into the objective function in (3.34), yielding

$$\begin{aligned} &\min_{P_1} \quad obj = \frac{K^2}{N} \cdot \left(\frac{1}{P_1} - 1 \right). \\ &s.t. \quad 1 > P_1 \geq P_f \end{aligned} \tag{3.45}$$

According to (3.43), K is equal to

$$K = \frac{\log\left(P_f\right)}{\log\left(P_1\right)}. \tag{3.46}$$

Substituting (3.46) into (3.45) yields

$$\begin{aligned} &\min_{P_1} \quad obj = \frac{\log^2\left(P_f\right)}{N} \cdot \frac{1}{\log^2\left(P_1\right)} \cdot \left(\frac{1}{P_1} - 1 \right) \Leftrightarrow \min_{P_1} \quad norm\_obj = \frac{1}{\log^2\left(P_1\right)} \cdot \left(\frac{1}{P_1} - 1 \right). \\ &s.t. \quad 1 > P_1 \geq P_f \qquad\qquad\qquad\qquad\qquad\qquad s.t. \quad 1 > P_1 \geq P_f \end{aligned} \tag{3.47}$$

To find the optimal value for P_1, we plot the normalized objective function value *norm_obj* in (3.47) as a function of P_1, as shown in Figure 3-2. From Figure 3-2, we can observe that the optimal value P_1^* is around 0.2

$$P_1^* \approx 0.2. \tag{3.48}$$

As mentioned earlier in this sub-section, P_1^* is not the optimal solution for the optimization problem in (3.27) due to the assumptions that we make. In reality, $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$ are statistically correlated which will be discussed in detail in Section 3.4. Considering the correlations between $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$, var[log($P_f^{SUS}$)] could be much larger than the objective function *obj* in (3.33). If the difference *diff* between var[log(P_f^{SUS})] and *obj* is large, P_1^* could be quite different from the optimal solution for (3.27). Regarding this, we should consider both *norm_obj* and *diff* when we choose P_1.

In this thesis, we empirically choose 0.1 as our target P_1 for two reasons

1) The *norm_obj* value at $P_1 = 0.1$ is very close to the optimal *norm_obj* value at $P_1 = P_1^*$.

2) The number of subsets given $P_1 = 0.1$ is smaller than that given $P_1 = P_1^*$, implying that the accumulated correlations between $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$ given $P_1 = 0.1$ is likely to be smaller than that given $P_1 = P_1^*$. Since the *norm_obj* values at $P_1 = 0.1$ and $P_1 = P_1^*$ are similar, less correlations results in a smaller var[log(P_f^{SUS})].

To sum up, we aim to select subsets that satisfy the following conditions

$$\begin{aligned} &P_k = P_{OBJ} \quad (k = 1, 2, \cdots, K-1) \\ &P_K \geq P_{OBJ} \\ &P_{OBJ} = 0.1 \end{aligned} \quad . \tag{3.49}$$

To this end, we propose to adaptively choose the failure events $\{F_k;\ k = 1, 2, \cdots, K\}$ to satisfy the following conditions

$$\begin{aligned} &P_k^{SUS} = P_{OBJ} \quad (k = 1, 2, \cdots, K-1) \\ &P_K^{SUS} \geq P_{OBJ} \\ &P_{OBJ} = 0.1 \end{aligned} \quad . \tag{3.50}$$

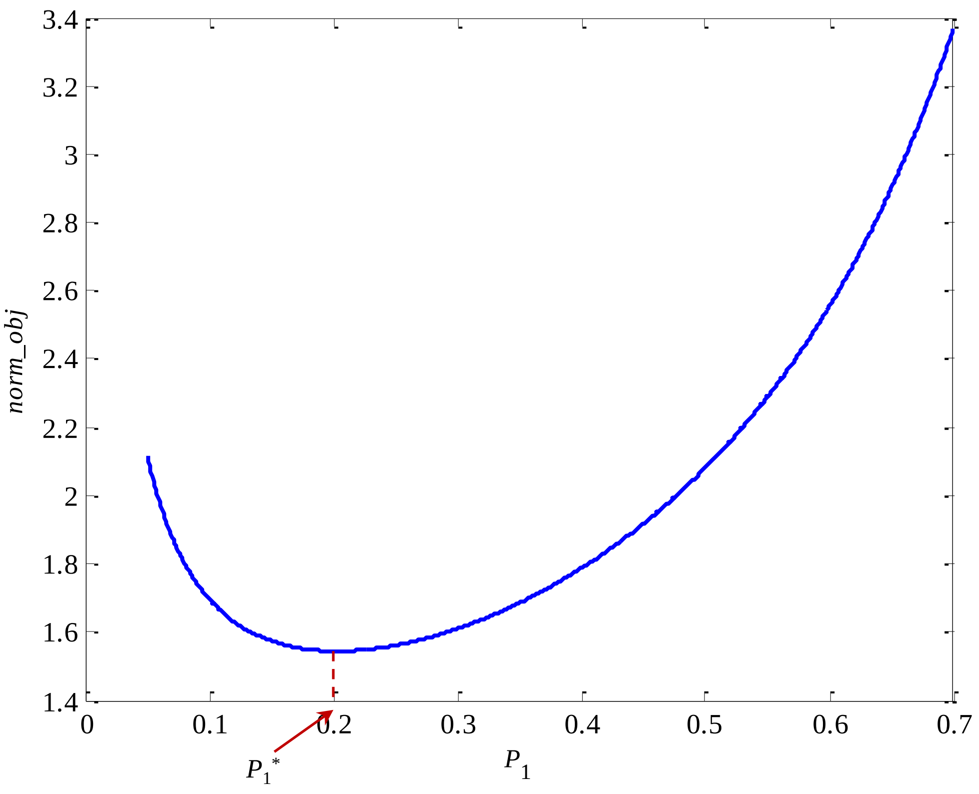

Figure 3-2. *norm_obj* in (3.47) is plotted as a function of P_1. The optimal value P_1^* corresponding to the minimum *norm_obj* is around 0.2.

To intuitively illustrate our proposed strategy for subset selection, we consider a sense amplifier example where the PoI $y(\mathbf{x})$ is the delay from the input to its output. If the delay $y(\mathbf{x})$ is greater than or equal to a pre-defined specification y_{SPEC}, we consider the sense amplifier as "FAIL". In this example, the failure events $\{F_k;\ k \in 1, 2, \cdots, K\}$ can be defined by setting different values for the delay specification. The delay specification should be tight (i.e., small) for F_1 and loose (i.e., large) for F_K. Our objective is to adaptively determine a set of monotonically increasing values $\{y_1 < y_2 < \cdots < y_{K-1} < y_K = y_{SPEC}\}$ and define the failure event F_k as $F_k = \{y \mid y(\mathbf{x}) \geq y_k\}$, where $k \in \{1, 2, \cdots, K\}$, so that the conditions in (3.50) are satisfied.

Based upon this idea, during the 1st phase of SUS for the sense amplifier example, we need to take the random samples $\{y[\mathbf{x}^{(1,\, l)}];\ l = 1, 2, \cdots, L_1\}$ created in Step 2 of Algorithm 3.1, and determine the value of y_1 so that the condition $y[\mathbf{x}^{(1,\, l)}] \geq y_1$ holds for $T_1 = L_1 \cdot P_1^{SUS} = L_1 \cdot P_{OBJ}$ samples. To this end, we sort $\{y[\mathbf{x}^{(1,\, l)}];\ l = 1, 2, \cdots, L_1\}$ and then set y_1 to the T_1-th largest value of y of the sorted samples. Similarly, during the 2nd phase, we need to take the random samples $\{\mathbf{x}^{(2,\, t,\, l)};\ t = 1, 2, \cdots, T_1,\ l = 1, 2, \cdots, L_2\}$ created in Step 6~16 of Algorithm 3.1, and determine the value of y_2 so that the condition $y[\mathbf{x}^{(2,\, t,\, l)}] \geq y_2$ holds for $T_2 = T_1 \cdot L_2 \cdot P_2^{SUS} = T_1 \cdot L_2 \cdot P_{OBJ}$ samples. We sort $\{\mathbf{x}^{(2,\, t,\, l)};\ t = 1, 2, \cdots, T_1,\ l = 1, 2, \cdots, L_2\}$ and then set y_2 to the T_2-th largest value of y of the sorted samples. The aforementioned procedure is repeatedly applied to further determine the values of $\{y_3, y_4, \cdots\}$ until we reach the K-th phase where setting $y_K = y_{SPEC}$ resulting in the probability P_K^{SUS} that is greater than P_{OBJ}.

Two important clarifications should be made for our proposed subset selection flow. First, combining (3.7), (3.21) and (3.50) implies

$$P_f \approx P_f^{SUS} = P_{OBJ}^{K-1} \cdot P_K^{SUS} \geq P_{OBJ}^{K}\,. \tag{3.51}$$

From (3.51), we observe that the total number of phases (i.e., K) depends on the failure rate P_f and P_{OBJ}. The value of K is approximately equal to the minimum integer that is greater than $\log(P_f) / \log(P_{OBJ})$. Since we do not know P_f in advance, we cannot pre-determine K. Instead, the value of K must be adaptively set when running the SUS algorithm.

Second, but more importantly, P_{OBJ} substantially impacts the efficiency of SUS. If P_{OBJ} is too small, the probabilities $\{P_k;\ k = 1, 2, \cdots, K\}$ are small. Hence, estimating these probabilities requires a large number of samples and, therefore, is not computationally efficient. On the other hand, if P_{OBJ} is too large, estimating $\{P_k;\ k = 1, 2, \cdots, K\}$ becomes trivial. However, based on (3.51), a large number of phases are needed to estimate the failure rate P_f (i.e., K is large), thereby resulting in high computational cost. For these reasons, it is crucial to appropriately choose P_{OBJ} to make SUS efficient for practical circuit analysis problems.

In this thesis, we set P_{OBJ} to 0.1, as shown in (3.50). In this case, even if the failure rate P_f is extremely small (e.g., 10^{-8}~10^{-6}), SUS only needs a small number of (e.g., K = 6~8) phases to complete. Furthermore, it only requires a few hundred samples during each phase to accurately estimate the probability P_k that is close to 0.1, where $k \in \{1, 2, \cdots, K\}$. It, in turn, results in an efficient implementation of SUS for rare failure rate estimation, as will be demonstrated by our experimental results in Section 3.5.

Next, let us discuss how to choose $\{N_k; k = 1, 2, \cdots, K\}$. From (3.44), we can see that $\{N_k; k = 1, 2, \cdots, K\}$ depend on the number of phases K and the total number of samples N that we would like to afford. As mentioned earlier, K is not aware in advance. Hence, we cannot determine the optimal $\{N_k; k = 1, 2, \cdots, K\}$ given a fixed N. In this thesis, we simply set $\{N_k; k = 1, 2, \cdots, K\}$ to a pre-defined value N_0

$$N_1 = N_2 = \cdots = N_K = N_0 . \tag{3.52}$$

As a result, the total number of samples $N = K \times N_0$ is determined online. Since our target probability P_{OBJ} is equal to 0.1, only a few hundred samples are sufficient to accurately estimate $\{P_k; k = 1, 2, \cdots, K\}$ that are close to 0.1. Therefore, we typically set N_0 to 10^2~10^3 in this thesis.

3.4 Confidence Interval Estimation

To quantitatively assess the accuracy of the proposed SUS estimator P_f^{SUS} shown in (3.21), we must estimate its confidence interval (CI). To this end, we need to know the distribution of P_f^{SUS}. Since P_f^{SUS} is equal to the multiplication of $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$, we should carefully study the statistical property of $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$ in order to derive the distribution for P_f^{SUS}.

As described in Algorithm 3.1, the estimators $\{P_k^{SUS}; k = 1, 2, \cdots, K\}$ are calculated from the random samples either drawn from $f(\mathbf{x})$ in Step 2 or created by MM in Step 6~16 of Algorithm 3.1. To be specific, P_1^{SUS} is calculated by using (3.10) with L_1 independent and identically distributed (i.i.d.) samples drawn from $f(\mathbf{x})$ in Step 2 of Algorithm 3.1. According to the central limit theorem (CLT) [71], P_1^{SUS} approximately follows a Normal distribution

$$P_1^{SUS} \sim N\left(P_1, v_1\right) \tag{3.53}$$

where the mean value P_1 is defined in (3.8) and the variance value v_1 can be approximated as [71]

$$v_1 \approx \frac{1}{L_1} \cdot P_1^{SUS} \cdot \left(1 - P_1^{SUS}\right) . \tag{3.54}$$

On the other hand, P_k^{SUS}, where $k \in \{2, 3, \cdots, K\}$, is calculated by using (3.26) with the MCMC samples $\{\mathbf{x}^{(k, t, l)}; t = 1, 2, \cdots, T_{k-1}, l = 1, 2, \cdots, L_k\}$ created by MM in Step 6~16 of Algorithm 3.1. It has been proved in [54] that all these MCMC samples $\{\mathbf{x}^{(k, t, l)}; t = 1, 2, \cdots, T_{k-1}, l = 1, 2, \cdots, L_k\}$ follow the conditional PDF $f(\mathbf{x} \mid \mathbf{x} \in \Omega_{k-1})$. Eq. (3.26) can be re-written as

$$P_k^{SUS} = \frac{1}{T_{k-1} \cdot L_k} \cdot \sum_{t=1}^{T_{k-1}} \sum_{l=1}^{L_k} I_{F_k}\left[y\left(\mathbf{x}^{(k,t,l)} \right) \right] \tag{3.55}$$

where $I_{Fk}[y(\mathbf{x})]$ represents the indicator function

$$I_{F_k}\left[y(\mathbf{x}) \right] = \begin{cases} 1 & y(\mathbf{x}) \in F_k \\ 0 & y(\mathbf{x}) \notin F_k \end{cases} . \tag{3.56}$$

From Step 7~15 of Algorithm 3.1, we can observe that $\{\mathbf{x}^{(k, t, l)}; l = 1, 2, \cdots, L_k\}$, where $t \in \{1, 2, \cdots, T_{k-1}\}$, are strongly correlated. Alternatively speaking, the MCMC samples $\{\mathbf{x}^{(k, t, l)}; t = 1, 2, \cdots, T_{k-1}, l = 1, 2, \cdots, L_k\}$ used to calculate P_k^{SUS} in (3.55) are not independent and, hence, cannot be considered as i.i.d. samples. For this reason, we cannot directly apply CLT [71] to derive the distribution for the estimator P_k^{SUS} in (3.55).

To address this issue, we define a set of new random variables

$$s^{(k,t)} = \frac{1}{L_k} \cdot \sum_{l=1}^{L_k} I_{F_k}\left[y\left(\mathbf{x}^{(k,t,l)} \right) \right] \quad \left(t = 1, 2, \cdots, T_{k-1} \right) . \tag{3.57}$$

Studying (3.57) reveals an important observation that $s^{(k, t)}$, where $t \in \{1, 2, \cdots, T_{k-1}\}$, only depends on the t-th Markov chain $\{\mathbf{x}^{(k, t, l)}; l = 1, 2, \cdots, L_k\}$. Since different Markov chains are created from different initial samples $\{\mathbf{x}^{(k, t, 1)}; t = 1, 2, \cdots, T_{k-1}\}$, the random variables $\{s^{(k, t)}; t = 1, 2, \cdots, T_{k-1}\}$ in (3.57) are almost statistically independent. Furthermore, since all the initial samples $\{\mathbf{x}^{(k, t, 1)}; t = 1, 2, \cdots, T_{k-1}\}$ follow the same conditional PDF $f(\mathbf{x} \mid \mathbf{x} \in \Omega_{k-1})$ and all the Markov chains are generated by following the same procedure (i.e., Step 7~15 of Algorithm 3.1), the random variables $\{s^{(k, t)}; t = 1, 2, \cdots, T_{k-1}\}$ should be identically distributed. For these reasons, we can consider $\{s^{(k, t)}; t = 1, 2, \cdots, T_{k-1}\}$ as a set of i.i.d. random variables.

Based on (3.57), P_k^{SUS} in (3.55) can be re-written as

$$P_k^{SUS} = \frac{1}{T_{k-1}} \cdot \sum_{t=1}^{T_{k-1}} s^{(k,t)} . \tag{3.58}$$

Since $\{s^{(k,\ t)};\ t = 1, 2, \cdots, T_{k-1}\}$ are i.i.d. random variables, P_k^{SUS} approximately follows a Normal distribution according to CLT

$$P_k^{SUS} \sim N\left[\mathrm{mean}\left(P_k^{SUS}\right), \mathrm{var}\left(P_k^{SUS}\right)\right] \tag{3.59}$$

where mean(•) and var(•) denote the mean and variance of a random variable respectively. From (3.55) and (3.56), mean(P_k^{SUS}) can be easily calculated as

$$\mathrm{mean}\left(P_k^{SUS}\right) = P_k \tag{3.60}$$

where P_k is defined in (3.9). Based on (3.58) where $\{s^{(k,\ t)};\ t = 1, 2, \cdots, T_{k-1}\}$ are i.i.d. random variables, var(P_k^{SUS}) can be computed as

$$\mathrm{var}\left(P_k^{SUS}\right) = \frac{1}{T_{k-1}^2} \cdot \sum_{t=1}^{T_{k-1}} \mathrm{VAR}\left[s^{(k,t)}\right] = \frac{1}{T_{k-1}} \cdot v_{s,k} \tag{3.61}$$

where $v_{s,k}$ represents the variance of the random variables $\{s^{(k,\ t)};\ t = 1, 2, \cdots, T_{k-1}\}$, and it can be approximated by the sample variance of $\{s^{(k,\ t)};\ t = 1, 2, \cdots, T_{k-1}\}$

$$v_{s,k} \approx \frac{1}{T_{k-1} - 1} \cdot \sum_{t=1}^{T_{k-1}} \left[s^{(k,t)} - P_k^{SUS}\right]^2 . \tag{3.62}$$

Substituting (3.60), (3.61) and (3.62) into (3.59) yields

$$P_k^{SUS} \sim N\left(P_k, v_k\right) \quad \left(k = 2, 3, \cdots, K\right) \tag{3.63}$$

where P_k is defined in (3.9) and

$$v_k \approx \frac{1}{\left(T_{k-1} - 1\right) \cdot T_{k-1}} \cdot \sum_{t=1}^{T_{k-1}} \left[s^{(k,t)} - P_k^{SUS}\right]^2 . \tag{3.64}$$

To further derive the distribution for P_f^{SUS} in (3.21) based on (3.53) and (3.63), we take logarithm on both sides of (3.21) and get (3.29). To derive the distribution of $\{\log(P_k^{SUS});\ k = 1, 2, \cdots, K\}$, we approximate the nonlinear function log(•) by the first-order Taylor expansion around the mean value P_k of the random variable P_k^{SUS}

$$\log\left(P_k^{SUS}\right) \approx \log\left(P_k\right) + \frac{P_k^{SUS} - P_k}{P_k} \approx \log P_k + \frac{P_k^{SUS} - P_k}{P_k^{SUS}} . \tag{3.65}$$

According to the linear approximation in (3.65), $\log(P_k^{SUS})$ follows a Normal distribution

$$\log\left(P_k^{SUS}\right) \sim N\left[\log\left(P_k\right), v_{\log,k}\right] \quad \left(k = 1, 2, \cdots, K\right) \tag{3.66}$$

where

$$v_{\log,k} = v_k \Big/ \left(P_k^{SUS}\right)^2 . \tag{3.67}$$

Since $\log(P_f^{SUS})$ is the summation of several “approximately” Normal random variables $\{\log(P_k^{SUS})$; $k = 1, 2, \cdots, K\}$, $\log(P_f^{SUS})$ also approximately follows a Normal distribution [71]

$$\log\left(P_f^{SUS}\right) \sim N\left\{\text{mean}\left[\log\left(P_f^{SUS}\right)\right], \text{var}\left[\log\left(P_f^{SUS}\right)\right]\right\} . \tag{3.68}$$

Based on (3.7), (3.29), and (3.66), $\text{mean}[\log(P_f^{SUS})]$ can be expressed as

$$\text{mean}\left[\log\left(P_f^{SUS}\right)\right] = \sum_{k=1}^{K} \log\left(P_k\right) = \log\left(\prod_{k=1}^{K} P_k\right) = \log\left(P_f\right) \tag{3.69}$$

and $\text{var}[\log(P_f^{SUS})]$ can be calculated as

$$\begin{aligned} \text{var}\left[\log\left(P_f^{SUS}\right)\right] &= \text{var}\left[\sum_{k=1}^{K} \log\left(P_k^{SUS}\right)\right] \\ &= \sum_{k=1}^{K} v_{\log,k} + 2 \cdot \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} \text{cov}\left[\log\left(P_i^{SUS}\right), \log\left(P_j^{SUS}\right)\right] \end{aligned} \tag{3.70}$$

where cov(•, •) denotes the covariance of two random variables.

When applying MCMC, we often observe that an MCMC sample is strongly correlated to its adjacent sample. However, the correlation quickly decreases as the distance between two MCMC samples increases. Therefore, we can assume that the samples used to estimate $\log(P_i^{SUS})$ are weakly correlated to the samples used to estimate $\log(P_j^{SUS})$, if the distance between i and j is greater than 1 (i.e., $|i - j| > 1$). Based on this assumption, Eq. (3.70) can be approximated as

$$\text{var}\left[\log\left(P_f^{SUS}\right)\right] \approx \sum_{k=1}^{K} v_{\log,k} + 2 \cdot \sum_{k=1}^{K-1} \text{cov}\left[\log\left(P_k^{SUS}\right), \log\left(P_{k+1}^{SUS}\right)\right] . \tag{3.71}$$

Accurately estimating the covariance between $\log(P_k^{SUS})$ and $\log(P_{k+1}^{SUS})$ is nontrivial. Here, we derive an upper bound for $\text{cov}[\log(P_k^{SUS}), \log(P_{k+1}^{SUS})]$ [71]

$$\text{cov}\left[\log\left(P_k^{SUS}\right), \log\left(P_{k+1}^{SUS}\right)\right] \le \sqrt{v_{\log,k} \cdot v_{\log,k+1}} \quad \left(k = 1, 2, \cdots, K-1\right) . \tag{3.72}$$

Substituting (3.72) into (3.71) yields

$$\text{var}\left[\log\left(P_f^{SUS}\right)\right] \le \sum_{k=1}^{K} v_{\log,k} + 2 \cdot \sum_{k=1}^{K-1} \sqrt{v_{\log,k} \cdot v_{\log,k+1}} = v_{\log,SUS} . \tag{3.73}$$

In this thesis, we approximate $\text{var}[\log(P_f^{SUS})]$ by its upper bound $v_{\log,SUS}$ defined in (3.73) to provide a conservative estimation for the CI. Based on (3.69) and (3.73), Eq. (3.68) can be re-written as

$$\log\left(P_f^{SUS}\right) \sim N\left[\log\left(P_f\right), v_{\log,SUS}\right]. \tag{3.74}$$

According to (3.74), we can derive the CI for any given confidence level. For instance, the 95% CI is expressed as

$$\left[\exp\left(\log\left(P_f^{SUS}\right) - 1.96\cdot\sqrt{v_{\log,SUS}}\right), \exp\left(\log\left(P_f^{SUS}\right) + 1.96\cdot\sqrt{v_{\log,SUS}}\right)\right]. \tag{3.75}$$

The aforementioned CI estimation accurately captures the uncertainty of our proposed estimator P_f^{SUS}, as will be demonstrated by numerical examples in the next sub-section.

3.5 Experimental Results

In this sub-section, we demonstrate the efficacy of SUS by an SRAM column example. Figure 3-3 shows the simplified schematic of an SRAM column consisting of 64 bit-cells designed in a 45nm CMOS process. In this example, our performance of interest is the read current I_{READ}, which is defined as the difference between two bit-line currents I_{BL} and I_{BLB} (i.e., $I_{READ} = I_{BL} - I_{BLB}$). If I_{READ} is too small, the voltage difference between *BL* and *BLB* is not large enough after pre-defined reading time and, hence, the value stored in bit-cell may not be correctly detected, resulting in read access failure. Because of this, if I_{READ} is greater than a pre-defined specification, we consider the SRAM circuit as "PASS". Otherwise, it is considered as "FAIL". Mathematically, the failure rate of the SRAM column is defined as

$$\Pr(\text{column fails}) = \Pr(\text{cell<0> fails } \textit{or} \text{ cell<1> fails } \textit{or} \ \cdots \ \textit{or} \text{ cell<63> fails}). \tag{3.76}$$

Eq. (3.76) is bounded by [71]

$$\Pr(\text{column fails}) \le \sum_{i=0}^{63} \Pr(\text{cell<}i\text{> fails}). \tag{3.77}$$

Since all the cells have the same design, we can assume that their failure rates under the worst-case scenario are the same. Based on this assumption, Eq. (3.77) can be re-written as

$$\Pr(\text{column fails}) \le 64\cdot P_f \tag{3.78}$$

where P_f denotes the cell failure rate under the worst-case scenario. The upper bound in (3.78) is a conservative bound, and can be quite larger than the SRAM column failure rate. How to accurately calculate the SRAM column failure rate from SRAM cell failure rates is still an open question, and is beyond the scope of this thesis. In this example, we will focus on estimating the cell failure rate P_f and then provide the upper bound for the column failure rate by using (3.78).

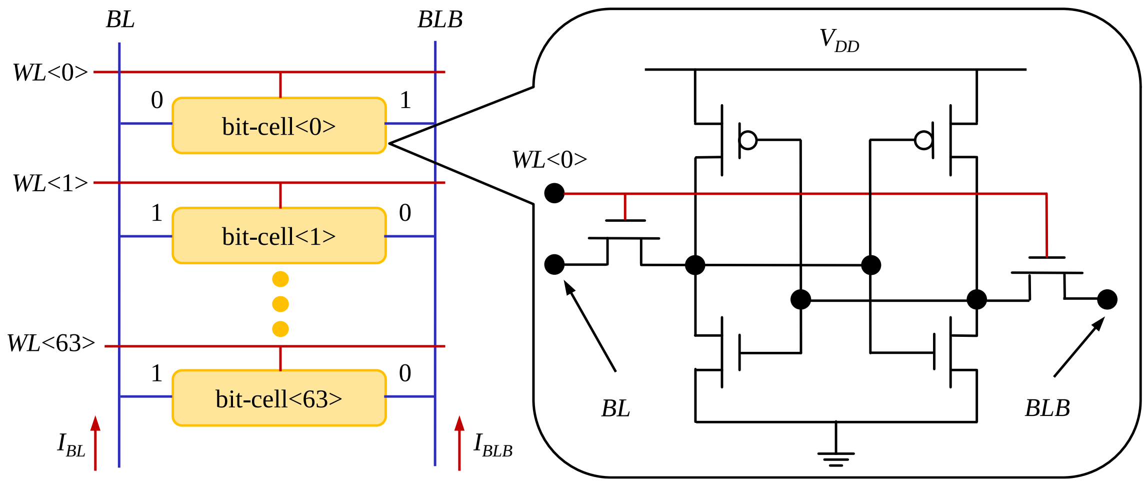

Figure 3-3. The simplified schematic is shown for an SRAM column consisting of 64 bit-cells designed in a 45nm CMOS process.

Without loss of generality, we estimate the failure rate of bit-cell<0> under the worst-case scenario. When reading bit-cell<0>, we first pre-charge both bit-lines to the supply voltage V_{DD}. Next, the word-line *WL*<0> is turned on and bit-cell<0> is activated. All other word-lines are turned off so that the corresponding bit-cells are de-activated. The current through bit-cell<0> then discharges the bit-lines and creates a voltage difference between *BL* and *BLB*. To mimic the worst-case scenario for read operation, we store "0" in bit-cell<0> and "1" in all other bit-cells. As such, the read current of bit-cell<0> discharges *BL*, while the leakage current of all other bit-cells discharges *BLB*, thereby slowing down the read operation.

To consider process variations in this experiment, we model the local V_{TH} mismatch of each transistor as an independent Normal random variable. Since the SRAM column consists of 64 bit-cells and each bit-cell is composed of 6 transistors, there are 384 transistors and, hence, 384 independent Normal random variables in total. It, in turn, serves as a good example to demonstrate the efficacy of SUS in a high-dimensional (i.e., M = 384) variation space.

Table 3-1. Failure rate P_f and 95% CI [P_f^L, P_f^U] estimated by MNIS and SUS ("golden" failure rate = 1.1×10^{-6})

# of Sims	MNIS			SUS		
	P_f^L	P_f	P_f^U	P_f^L	P_f	P_f^U
3600	0	9.2×10^{-11}	2.5×10^{-10}	2.1×10^{-7}	1.2×10^{-6}	7.4×10^{-6}
4200	0	6.8×10^{-11}	1.8×10^{-10}	1.2×10^{-7}	7.6×10^{-7}	4.7×10^{-6}
4800	0	5.4×10^{-11}	1.4×10^{-10}	1.9×10^{-7}	9.9×10^{-7}	5.1×10^{-6}
5400	0	2.1×10^{-10}	4.7×10^{-10}	1.7×10^{-7}	8.9×10^{-7}	4.7×10^{-6}
6000	0	1.8×10^{-10}	4.0×10^{-10}	2.3×10^{-7}	1.0×10^{-6}	4.5×10^{-6}

For testing and comparison purposes, three different algorithms are implemented: (i) brute-force MC, (ii) minimum-norm importance sampling (MNIS) [37], and (iii) SUS. In our experiments, the brute-force MC is applied to generate the "golden" failure rate which is used to evaluate the accuracy of the other two approaches. As described in [37], MNIS consists of two stages: (i) 2000 transistor-level simulations are first used to search the variation space, and (ii) a shifted Normal PDF is then constructed to perform importance sampling and estimate the rare failure rate.

We first run the brute-force MC approach with 10^9 samples. The estimated failure rate by MC is 1.1×10^{-6}, which is considered as the "golden" failure rate in this example. Table 3-1 summarizes the failure rate and the 95% CI estimated by MNIS and SUS with different numbers of simulations. From Table 3-1, we have two important observations.

First, the estimated failure rate from MNIS is strongly biased. We believe that the shifted Normal PDF used by MNIS for importance sampling does not capture the most important failure region and, hence, the estimated failure rate is substantially less than the "golden" failure rate. More importantly, the 95% CI estimated by MNIS cannot cover the "golden" failure rate, implying that MNIS also fails to assess the accuracy of its estimated failure rate. This is an important limitation of MNIS since the user cannot reliably know the actual "confidence" of the estimator in practice.

Second, for our proposed SUS method, the estimator is almost unbiased. The 95% CI is tight, and can accurately cover the "golden" failure rate. It, in turn, demonstrates that SUS can achieve substantially better accuracy than the traditional method (i.e., MNIS) in a high-dimensional variation space.

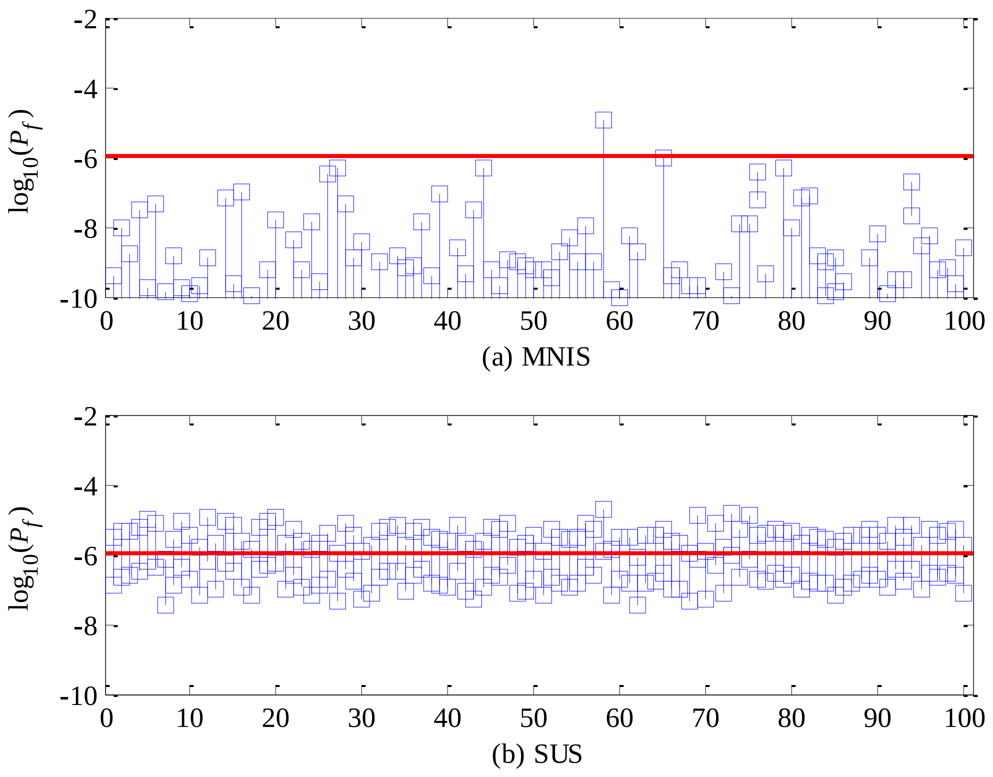

Figure 3-4. The 95% CIs are estimated from 100 repeated runs with 6000 simulations in each run for (a) MNIS and (b) SUS. The red line represents the "golden" failure rate.

Next, we further verify the accuracy of the 95% CI estimated by MNIS and SUS. To this end, we repeatedly run each method for 100 times with 6000 simulations in each run. Figure 3-4 shows the 100 estimated 95% CIs for each method, where each blue bar represents the CI of a single run, and the red line represents the "golden" failure rate. To clearly plot these CIs, the y-axis of Figure 3-4 is displayed in logarithmic scale over the range [10^{-10}, 10^{-2}]. If this range cannot cover a complete CI, we only show a portion of the CI that is inside the range.

Studying Figure 3-4 reveals several important observations. First, only a single CI estimated from 100 repeated runs by MNIS can cover the "golden" failure rate. It, again, demonstrates the fact that the confidence level of MNIS cannot be accurately assessed by its estimated CI. Second, there are 95 CIs out of 100 runs that cover the "golden" failure rate for SUS. It demonstrates that the CIs estimated by SUS accurately measure the estimation error and can appropriately indicate the conference level of the estimator.

Once we know the cell failure rate P_f, we can calculate the upper bound for the column failure rate by using (3.78).

3.6 Algorithm Limitations

Similar to most sampling approaches, SUS cannot accurately estimate our interested failure rate if it does not capture all the important failure regions. There are two typical scenarios where SUS may fail to work:

- **Scenario 1**: there are a large number of disjoint failure regions in the variation space and it is very unlikely that SUS can cover all these failure regions with only a few hundred or thousand samples. To clearly understand this scenario, let us consider one extreme case where the number of disjoint failure regions exponentially increases with the dimensionality. The failure region is defined as follows:

$$\begin{gathered}\Omega = \left\{ \mathbf{x} \left| \sum_{m=1}^{M} \left(x_m - c_m \right)^2 < 0.25 \right. \right\} \\ c_m \in \{-1, 1\} \quad m = 1, 2, \cdots, M\end{gathered} \tag{3.79}$$

  where $\{c_m; m = 1, 2, \cdots, M\}$ can take either -1 or 1, resulting in 2^M combinations of $[c_1\ c_2 \cdots c_M]$. Hence, the failure region Ω consists of 2^M disjoint M-balls. For illustration purposes, a 2-dimensional example with the failure region defined in (3.79) is shown in Figure 3-5.

  In order to cover all the disjoint failure regions, the required number of samples will exponentially increase with the dimensionality. In a high-dimensional space, the required number of samples is so large that we cannot afford it. In reality, SUS only generates a few hundred or thousand samples and, hence, unavoidably misses many important failure regions in this extreme case.

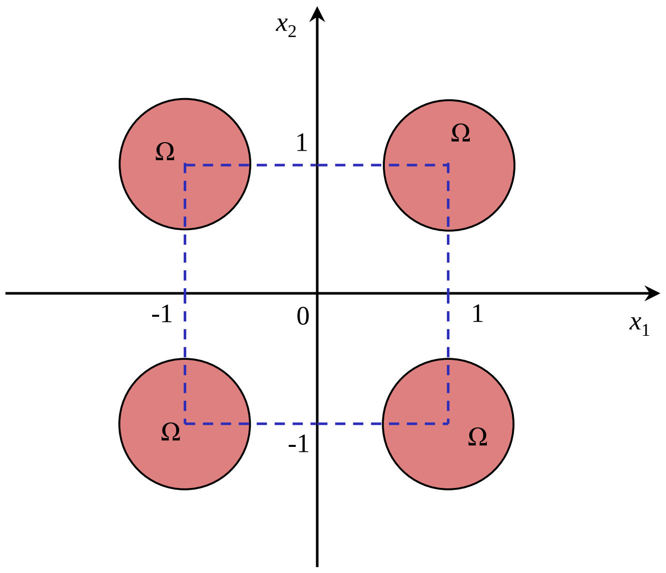

Figure 3-5. A 2-dimensional example is used to illustrate the extreme case of scenario 1.

- **Scenario 2**: the unimportant region at lower phases becomes important at higher phases. To clearly understand this scenario, let us consider a simple 1-dimensional example, as shown in Figure 3-6. The blue curve in Figure 3-6 (a) denotes the Normal PDF $f(\mathbf{x})$ for $\mathbf{x}$. In Figure 3-6 (b), the green curve denotes the performance function $y(\mathbf{x})$. If the performance of interest y is smaller than our pre-defined specification y_{spec}, the circuit is considered as "FAIL". In this example, the failure region $\boldsymbol{\Omega}$ consists of two parts (i.e., two red color regions): one part is on the left side and the other part is on the right side. Since $\mathbf{x}$ follows a Normal PDF, the probability mass associated with the left failure region is larger than that associated with the right failure region. Alternatively speaking, the left failure region is more important than the right failure region in this example.

  For this simple 1-dimensional example, SUS will adaptively determine a set of monotonically decreasing values $\{y_1 > y_2 > \cdots > y_{K-1} > y_K = y_{SPEC}\}$ and define the failure event F_k as $F_k = \{y \mid y(\mathbf{x}) \leq y_k\}$, where $k \in \{1, 2, \cdots, K\}$. In the first phase, SUS generates a number of samples from $f(\mathbf{x})$, and determines y_1 based on the performance values of these samples, as shown in Figure 3-6 (c). Yellow points denote samples that do not belong to F_1, and green points denote samples that belong to F_1. In this example, only the right part is considered as the important region in the first phase, and SUS will only explore the right part in the following phases, as shown in Figure 3-6 (d). Restating in words, because the left part is not considered as an important region at lower phases (i.e., the first phase), the most important failure region (i.e., the left red color region in Figure 3-6 (b)) sitting on the left side is not captured by SUS, leading to a significant biased estimation.

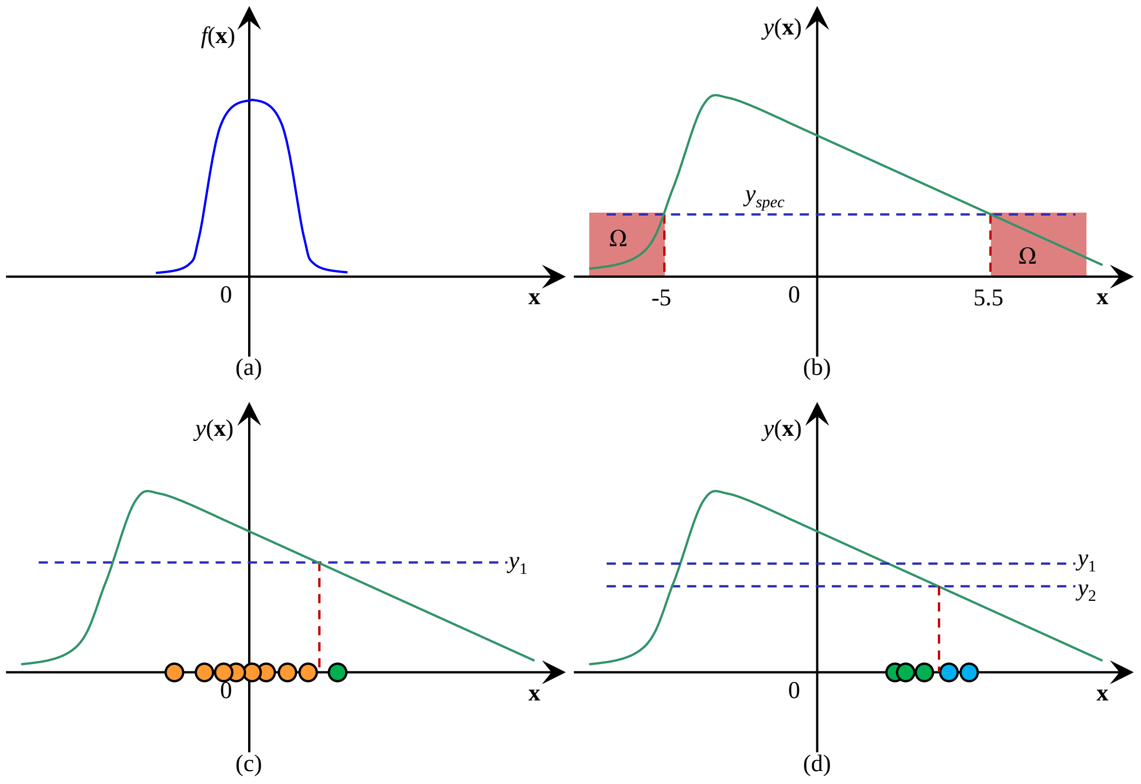

Figure 3-6. A 1-dimensional example is used to illustrate scenario 2. (a) **x** follows a Normal PDF $f(\mathbf{x})$. (b) Green curve denotes the performance function, and two red color regions denote failure regions. (c) Draw random samples from $f(\mathbf{x})$ and determine the intermediate failure event F_1 as $F_1 = \{y \mid y(\mathbf{x}) \le y_1\}$. Yellow points denote samples that do not belong to F_1, and green points denote samples that belong to F_1. (d) Draw random samples from $f(\mathbf{x} \mid y(\mathbf{x}) \le y_1)$ by MM and determine the intermediate failure event F_2 as $F_2 = \{y \mid y(\mathbf{x}) \le y_2\}$. Green points denote samples that do not belong to F_2, and blue points denote samples that belong to F_2.

Estimation failures under the aforementioned scenarios are difficult to detect. Though these scenarios can theoretically occur, we rarely see them in the real test cases. Hence, we can safely apply SUS for most circuit designs.

To define the intermediate failure events required by SUS, the circuit performance of interest should be continuous. However, the performances of interest in the SRAM circuits can be discrete. Take voltage-mode sense amplifier as an example. When studying the stability of the sense amplifier, the performance of interest is the binary output of the sense amplifier. To estimate rare failure events for discrete performances of interest in a high-dimensional variation space, we further propose a scaled-sigma sampling (SSS) approach. In the next section, we will describe our proposed SSS approach in detail.

Chapter 4

Scaled-Sigma Sampling for Discrete Performance Metrics

As mentioned in Chapter 3, SUS can only estimate the rare failure events for circuits that have continuous performances of interest. To address this limitation and estimate the rare failure probability for discrete performances of interest in a high-dimensional variation space, *scaled-sigma sampling* (SSS) is proposed in this chapter. The key idea of SSS is to generate "scaled" random samples from the "scaled" PDFs and estimate the "scaled" failure rates based on these "scaled" random samples. Here, the "scaled" PDF refers to the distorted PDF for which the standard deviation of the original PDF is scaled up. By scaling up the standard deviation, the rare failure event associated with the original PDF becomes not-so-rare in the "scaled" PDFs and, therefore, the "scaled" failure rates corresponding to the "scaled" PDFs can be efficiently estimated.

To further recover the original rare failure rate from the scaled failure rates, we derive an analytical model between the scaled failure rate and the scaling factor based on the theorem of "soft maximum" [73], which is the key contribution of SSS work. To fit the model, we first choose a set of scaling factors, and estimate their corresponding scaled failure rates from a small number of scaled random samples. The model is then optimally fitted by applying maximum-likelihood estimation (MLE) [72]. Next, the original rare failure rate can be efficiently estimated from the fitted model.

Unlike the traditional estimator where a statistical metric is estimated by the average of multiple random samples and, hence, the confidence interval can be derived as a closed-form expression, our proposed SSS estimator is calculated by linear regression with nonlinear exponential/logarithmic transformation, as shown in Section 4.3. Therefore, accurately estimating the confidence interval of SSS is not a trivial task. To address this challenge, we apply bootstrap technique [70]. The key idea of bootstrap is to re-generate a large number of random samples based on a statistical model without running additional transistor-level simulations. These random samples are then used to repeatedly calculate the failure rate by SSS for multiple times. Based on these repeated runs, the statistics (hence, the confidence interval) of the proposed SSS estimator can be accurately estimated.

In a commercial process design kit, the random variables $\{x_m; m = 1, 2, \cdots, M\}$ in the vector $\mathbf{x}$ shown in (2.13) are typically modeled as a set of independent random variables following Gaussian and/or uniform distributions. In Section 4.1, we describe our proposed scaled-sigma sampling (SSS) approach for Gaussian distribution. Namely, the random variable $\mathbf{x}$ follows a multivariate Gaussian distribution. In Section 4.2, we will further extend SSS to "Gaussian-Uniform" distribution where the random variables $\{x_m; m = 1, 2, \cdots, M\}$ in (2.13) are mutually independent, a subset of these random variables follow the standard Gaussian distributions and the other random variables are uniformly distributed. Such a multivariate Gaussian-Uniform distribution has been used in many process design kits today.

4.1 SSS for Gaussian Random Variables

Without loss of generality, we assume that the random variables $\{x_m; m = 1, 2, \cdots, M\}$ in the vector $\mathbf{x}$ are mutually independent and follow standard Gaussian distributions, as shown in (2.23). Unlike the traditional importance sampling methods that must explicitly identify the high-probability failure region, SSS takes a completely different strategy to address the following two fundamental questions:

1) How to efficiently draw random samples from the high-probability failure region?
2) How to estimate the failure rate based on these random samples?

In what follows, we will derive the mathematical formulation of SSS for Gaussian distribution and highlight its novelties.

4.1.1 Statistical Sampling

For the application of rare failure rate estimation, $f(\mathbf{x})$ in (2.23) is often extremely small for a random sample $\mathbf{x}$ inside the failure region. It implies that the failure region is far away from the origin $\mathbf{x} = \mathbf{0}$, as shown in Figure 4-1 (a). Since the failure rate is extremely small, the brute-force Monte Carlo approach cannot efficiently draw random samples from the failure region. Namely, many MC samples cannot reach the tail of $f(\mathbf{x})$.

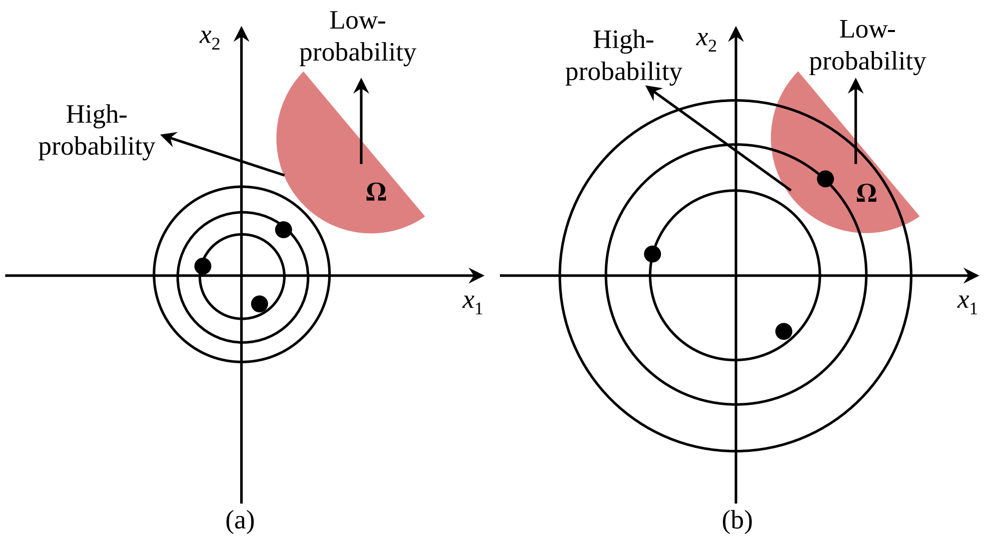

Figure 4-1. The proposed SSS method for Gaussian distribution is illustrated by a 2-D example where the red area Ω denotes the failure region and the circles represent the contour lines of the PDF. (a) Rare failure events occur at the tail of the original PDF $f(\mathbf{x})$ and the failure region is far away from the origin $\mathbf{x} = \mathbf{0}$. (b) The scaled PDF $g(\mathbf{x})$ widely spreads over a large region and the scaled samples are likely to reach the faraway failure region.

In this thesis, we apply a simple idea to address the aforementioned sampling issue. Given $f(\mathbf{x})$ in (2.23) for the M-dimensional random variable $\mathbf{x}$, we scale up the standard deviation of $\mathbf{x}$ by a scaling factor s ($s > 1$), yielding the following distribution

$$g(\mathbf{x}) = \prod_{m=1}^{M} \left[\frac{1}{\sqrt{2\pi} \cdot s} \cdot \exp\left(-\frac{x_m^2}{2s^2} \right) \right] = \frac{\exp\left(-\|\mathbf{x}\|_2^2 / 2s^2 \right)}{\left(\sqrt{2\pi}\right)^M \cdot s^M} . \tag{4.1}$$

Once the standard deviation of $\mathbf{x}$ is increased by a factor of s, we conceptually increase the magnitude of process variations. Hence, the scaled PDF $g(\mathbf{x})$ widely spreads over a large region and the probability for a random sample to reach the faraway failure region increases, as shown in Figure 4-1 (b).

From an alternative viewpoint, the original random variables $\{x_m; m = 1, 2, \cdots, M\}$ follow the independent standard Gaussian distributions defined in (2.23). If we scale each of them (say, x_m) by a factor of s, the scaled random variables $\{s \cdot x_m;\ m = 1, 2, \cdots, M\}$ follow $g(\mathbf{x})$ in (4.1). Therefore, when sampling $g(\mathbf{x})$, we can first draw random samples from $f(\mathbf{x})$ and then scale each random sample by a factor of s. As a result, the scaled samples will move far away from the origin $\mathbf{x} = \mathbf{0}$ and are likely to reach the failure region, as shown in Figure 4-1 (b).

On the other hand, it is important to note that the mean of $g(\mathbf{x})$ remains $\mathbf{0}$, which is identical to the mean of $f(\mathbf{x})$. Hence, for a given sampling location $\mathbf{x}$, the likelihood defined by $g(\mathbf{x})$ remains inversely proportional to the length of the vector $\mathbf{x}$ (i.e., $||\mathbf{x}||_2$). Namely, it is more (or less) likely to reach the sampling location $\mathbf{x}$, if the distance between the location $\mathbf{x}$ and the origin $\mathbf{0}$ is smaller (or larger). It, in turn, implies that the high-probability failure region associated with $f(\mathbf{x})$ remains the high-probability failure region after the PDF is scaled to $g(\mathbf{x})$, as shown in Figure 4-1 (a) and (b). Scaling the PDF from $f(\mathbf{x})$ to $g(\mathbf{x})$ does not change the location of the high-probability failure region. Instead, it only makes the failure region easy to sample.

Once the scaled random samples are drawn from $g(\mathbf{x})$ in (4.1), we need to further estimate P_f defined in (2.14). Since $g(\mathbf{x})$ and $f(\mathbf{x})$ are different, we cannot simply average the random samples generated by $g(\mathbf{x})$ to calculate P_f defined by $f(\mathbf{x})$. A major contribution of SSS is to derive an analytical model to accurately estimate the failure rate P_f from the scaled random samples, as will be discussed in detail in the next sub-section.

4.1.2 Failure Rate Estimation

Given N random samples $\{\mathbf{x}^{(n)};\ n = 1, 2, \cdots, N\}$ drawn from $g(\mathbf{x})$ in (4.1), one straightforward way to estimate P_f is based upon the theory of importance sampling. Namely, since the random samples are generated by $g(\mathbf{x})$ that is different from $f(\mathbf{x})$, we can estimate the failure rate P_f by calculating the average of $f(\mathbf{x}) \cdot I(\mathbf{x}) / g(\mathbf{x})$, as shown by the estimator P_f^{IS} in (2.20).

Such a simple approach, however, does not result in an accurate failure rate, if the dimensionality of the variation space (i.e., M) is large. To understand the reason, let us calculate the variance of the importance sampling estimator P_f^{IS} in (2.20)

$$\mathrm{var}\left(P_f^{IS}\right)=\mathrm{var}\left\{\frac{1}{N}\cdot\sum_{n=1}^{N}f\left[\mathbf{x}^{(n)}\right]\cdot I\left[\mathbf{x}^{(n)}\right]\Big/g\left[\mathbf{x}^{(n)}\right]\right\}. \tag{4.2}$$

Since $\{\mathbf{x}^{(n)}; n = 1, 2, \cdots, N\}$ are independently drawn from $g(\mathbf{x})$ in (4.1), Eq. (4.2) can be re-written as

$$\mathrm{var}\left(P_f^{IS}\right)=\frac{1}{N}\cdot\mathrm{var}\left[\frac{f(\mathbf{x})\cdot I(\mathbf{x})}{g(\mathbf{x})}\right]. \tag{4.3}$$

In (4.3), $\mathrm{var}[f(\mathbf{x})\cdot I(\mathbf{x})/g(\mathbf{x})]$ can be expressed as [71]

$$\mathrm{var}\left[\frac{f(\mathbf{x})\cdot I(\mathbf{x})}{g(\mathbf{x})}\right]=\mathrm{E}\left\{\left[\frac{f(\mathbf{x})\cdot I(\mathbf{x})}{g(\mathbf{x})}\right]^2\right\}-\left\{\mathrm{E}\left[\frac{f(\mathbf{x})\cdot I(\mathbf{x})}{g(\mathbf{x})}\right]\right\}^2 \tag{4.4}$$

where E(•) denotes the expected value of a random variable. We can re-write (4.4) as

$$\begin{aligned}\mathrm{var}\left[\frac{f(\mathbf{x})\cdot I(\mathbf{x})}{g(\mathbf{x})}\right]&=\int_{-\infty}^{+\infty}\frac{f^2(\mathbf{x})}{g^2(\mathbf{x})}\cdot I^2(\mathbf{x})\cdot g(\mathbf{x})\cdot d\mathbf{x}-\left[\int_{-\infty}^{+\infty}\frac{f(\mathbf{x})}{g(\mathbf{x})}\cdot I(\mathbf{x})\cdot g(\mathbf{x})\cdot d\mathbf{x}\right]^2\\&=\int_{-\infty}^{+\infty}\frac{f^2(\mathbf{x})}{g(\mathbf{x})}\cdot I(\mathbf{x})\cdot d\mathbf{x}-\left[\int_{-\infty}^{+\infty}f(\mathbf{x})\cdot I(\mathbf{x})\cdot d\mathbf{x}\right]^2\end{aligned}. \tag{4.5}$$

Based on (2.15), Eq. (4.5) can be simplified as

$$\mathrm{var}\left[\frac{f(\mathbf{x})\cdot I(\mathbf{x})}{g(\mathbf{x})}\right]=\int_{-\infty}^{+\infty}\frac{f^2(\mathbf{x})}{g(\mathbf{x})}\cdot I(\mathbf{x})\cdot d\mathbf{x}-P_f^2. \tag{4.6}$$

Since $I(\mathbf{x})$ is less than or equal to 1, Eq. (4.6) is bounded by

$$\mathrm{var}\left[\frac{f(\mathbf{x})\cdot I(\mathbf{x})}{g(\mathbf{x})}\right]\le\int_{-\infty}^{+\infty}\frac{f^2(\mathbf{x})}{g(\mathbf{x})}\cdot d\mathbf{x}-P_f^2. \tag{4.7}$$

At the right-hand side of (4.7), P_f is the failure rate and, hence, the second term (i.e., P_f^2) is a constant. Based on (2.23) and (4.1), the first term at the right-hand side of (4.7) can be calculated as

$$\begin{aligned}\int_{-\infty}^{+\infty}\frac{f^2(\mathbf{x})}{g(\mathbf{x})}\cdot d\mathbf{x}&=\frac{s^M}{\left(\sqrt{2\cdot\pi}\right)^M}\cdot\int_{-\infty}^{+\infty}\exp\left[-\frac{\left(2\cdot s^2-1\right)}{2\cdot s^2}\cdot\mathbf{x}^2\right]\cdot d\mathbf{x}\\&=\left[\frac{s^4}{2\cdot s^2-1}\right]^{\frac{M}{2}}\end{aligned}. \tag{4.8}$$

Based on (4.3), (4.7) and (4.8), we have

$$\mathrm{var}\left(P_f^{IS}\right) \le \frac{1}{N}\cdot\left\{\left[\frac{s^4}{2\cdot s^2-1}\right]^{\frac{M}{2}} - P_f^2\right\}. \tag{4.9}$$

Eq. (4.9) shows that the upper bound of the variance of the estimator P_f^{IS} exponentially increases with the dimensionality M when s is greater than 1. It, in turn, implies that the variance of P_f^{IS} can be prohibitively large in a high-dimensional variation space. It is equivalent to saying that the estimator P_f^{IS} based on importance sampling may not be sufficiently accurate when the variation space is high-dimensional. It does not fit our need of high-dimensional failure rate estimation in this thesis.

Instead of relying on the theory of importance sampling, our proposed SSS method attempts to estimate the failure rate P_f from a completely different avenue. We first take a look at the "scaled" failure rate corresponding to the scaled PDF $g(\mathbf{x})$

$$P_g = \int_{-\infty}^{+\infty} I(\mathbf{x})\cdot g(\mathbf{x})\cdot d\mathbf{x}. \tag{4.10}$$

Our objective is to study the relation between the scaled failure rate P_g in (4.10) and the original failure rate P_f in (2.15). Towards this goal, we partition the M-dimensional variation space into a large number of identical hyper-rectangles with the same volume and the scaled failure rate P_g in (4.10) can be approximated as

$$P_g \approx \sum_k I\left[\mathbf{x}^{(k)}\right]\cdot g\left[\mathbf{x}^{(k)}\right]\cdot\Delta\mathbf{x} \tag{4.11}$$

where $\mathbf{x}^{(k)}$ represents the center of the k-th hyper-rectangle, and $\Delta\mathbf{x}$ denotes the volume of a hyper-rectangle. The approximation in (4.11) is accurate, if each hyper-rectangle is sufficiently small. Given (2.16), Eq. (4.11) can be re-written as

$$P_g \approx \sum_{k\in\Omega} g\left[\mathbf{x}^{(k)}\right]\cdot\Delta\mathbf{x} \tag{4.12}$$

where $\{k;\, k \in \Omega\}$ represents the set of all hyper-rectangles that fall into the failure region. Substituting (4.1) into (4.12), we have

$$P_g \approx \frac{\Delta\mathbf{x}}{\left(\sqrt{2\pi}\right)^M\cdot s^M}\cdot\sum_{k\in\Omega}\exp\left[-\left\|\mathbf{x}^{(k)}\right\|_2^2 \Big/ 2s^2\right]. \tag{4.13}$$

Taking the logarithm on both sides of (4.13) yields

$$\log P_g \approx \log \frac{\Delta \mathbf{x}}{\left(\sqrt{2\pi}\right)^M} - M \cdot \log s + \operatorname*{lse}_{k \in \Omega}\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] \tag{4.14}$$

where

$$\operatorname*{lse}_{k \in \Omega}\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] = \log\left\{ \sum_{k \in \Omega} \exp\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] \right\} \tag{4.15}$$

stands for the log-sum-exp function. The function lse(•) in (4.15) is also known as the "soft maximum" from the mathematics [73]. It can be bounded by

$$\max_{k \in \Omega}\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] \le \operatorname*{lse}_{k \in \Omega}\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] \le \max_{k \in \Omega}\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] + \log(T) \tag{4.16}$$

where T denotes the total number of hyper-rectangles in Ω.

To clearly understand (4.16), let us consider two extreme cases. First, let us assume that all the hyper-rectangles $\{\mathbf{x}^{(k)};\ k \in \Omega\}$ have the same distance to the origin **0**. In this case, the function lse(•) reaches its upper bound in (4.16). Second, we assume that only one hyper-rectangle in the set $\{\mathbf{x}^{(k)};\ k \in \Omega\}$ is close to the origin **0**, and all other hyper-rectangles are far away from the origin **0**. In this case, the function lse(•) reaches its lower bound in (4.16). For our application of rare failure event analysis, however, these two ideal cases rarely occur. Therefore, we cannot simply use the lower or upper bound in (4.16) to approximate the function lse(•) in (4.15).

In general, there exist a number of (say, T_0) *dominant* hyper-rectangles that are much closer to the origin **0** than other hyper-rectangles in the set $\{\mathbf{x}^{(k)};\ k \in \Omega\}$. Without loss of generality, we assume that the first T_0 hyper-rectangles $\{\mathbf{x}^{(k)};\ k = 1, 2, \cdots, T_0\}$ are dominant. Hence, we can approximate the function lse(•) in (4.15) as

$$\operatorname*{lse}_{k \in \Omega}\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] \approx \log\left\{ \sum_{k=1}^{T_0} \exp\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] \right\}. \tag{4.17}$$

We further assume that these dominant hyper-rectangles $\{\mathbf{x}^{(k)};\ k = 1, 2, \cdots, T_0\}$ have similar distances to the origin **0**. Thus, Eq. (4.17) can be approximated by

$$\operatorname*{lse}_{k \in \Omega}\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] \approx \max_{k \in \Omega}\left[-\left\| \mathbf{x}^{(k)} \right\|_2^2 \Big/ 2s^2 \right] + \log(T_0). \tag{4.18}$$

Substituting (4.18) into (4.14) yields

$$\log P_g \approx \alpha + \beta \cdot \log s + \frac{\gamma}{s^2} \tag{4.19}$$

where

$$\alpha = \log\left[\Delta\mathbf{x} \middle/ \left(\sqrt{2\pi}\right)^M \right] + \log\left(T_0\right) \tag{4.20}$$

$$\beta = -M \tag{4.21}$$

$$\gamma = -\frac{1}{2} \cdot \min_{k\in\Omega}\left[\left\| \mathbf{x}^{(k)} \right\|_2^2 \right]. \tag{4.22}$$

Eq. (4.19) reveals the important relation between the scaled failure rate P_g and the scaling factor s. The approximation in (4.19) does not rely on any specific assumption of the failure region. It is valid, even if the failure region is non-convex or discontinuous.

While (4.20)-(4.22) show the theoretical definition of the model coefficients α, β and γ, finding their exact values is not trivial. For instance, the coefficient γ is determined by the hyper-rectangle that falls into the failure region Ω and is closest to the origin $\mathbf{x} = \mathbf{0}$. In practice, without knowing the failure region Ω, we cannot directly find out the value of γ. For this reason, we propose to determine the analytical model in (4.19) by linear regression. Namely, we first estimate the scaled failure rates $\{P_{g,q}; q = 1, 2, \cdots, Q\}$ by setting the scaling factor s to a number of different values $\{s_q; q = 1, 2, \cdots, Q\}$. As long as the scaling factors $\{s_q; q = 1, 2, \cdots, Q\}$ are sufficiently large, the scaled failure rates $\{P_{g,q}; q = 1, 2, \cdots, Q\}$ are large and can be accurately estimated with a small number of random samples. Next, the model coefficients α, β and γ are fitted by linear regression for the model template in (4.19) based on the values of $\{(s_q, P_{g,q}); q = 1, 2, \cdots, Q\}$. Once α, β and γ are known, the original failure rate P_f in (2.15) can be predicted by *extrapolation*. Namely, we substitute $s = 1$ into the analytical model in (4.19)

$$\log P_f^{SSS} = \alpha + \gamma \tag{4.23}$$

where P_f^{SSS} denotes the value of P_f estimated by SSS. Apply the exponential function to both sides of (4.23) and we have

$$P_f^{SSS} = \exp\left(\alpha + \gamma\right). \tag{4.24}$$

While the aforementioned discussions reveal the theoretical framework of the proposed SSS method, a number of implementation issues must be carefully studied to make SSS of practical utility. These implementation details will be further discussed in Section 4.3.1-4.3.3.

4.2 SSS for Gaussian-Uniform Random Variables

In Section 4.1, we assume that all random variables in **x** are mutually independent and follow the standard Gaussian distributions after applying principal component analysis. Such an assumption, however, may not always hold for today's nanoscale IC technologies. Namely, $\{x_m;\ m = 1, 2, \cdots, M\}$ may be modeled as other probability distributions (e.g., uniform distribution, etc.) in many practical applications.

In this section, we further extend SSS to Gaussian-Uniform distribution. Without loss of generality, we re-write **x** as

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_G \\ \mathbf{x}_U \end{bmatrix} \tag{4.25}$$

where the vector $\mathbf{x}_G = [x_{G,1}\ x_{G,2}\ \cdots\ x_{G,MG}]^T$ includes M_G random variables following the standard Gaussian distributions, and the vector $\mathbf{x}_U = [x_{U,1}\ x_{U,2}\ \cdots\ x_{U,MU}]^T$ includes M_U random variables following the uniform distributions, and $M = M_G + M_U$ is the total number of these random variables. Since all random variables in the vector **x** are mutually independent, we can express the joint PDF as

$$f(\mathbf{x}) = f_g(\mathbf{x}_G) \cdot f_u(\mathbf{x}_U) \tag{4.26}$$

$$f_g(\mathbf{x}_G) = \prod_{m=1}^{M_G} \left[\frac{1}{\sqrt{2\pi}} \cdot \exp\left(-\frac{x_{G,m}^2}{2} \right) \right] = \frac{\exp\left(-\|\mathbf{x}_G\|_2^2 / 2 \right)}{\left(\sqrt{2\pi} \right)^{M_G}} \tag{4.27}$$

$$f_u(\mathbf{x}_U) = \prod_{m=1}^{M_U} \left(\frac{1}{u_m - l_m} \right) \cdot I\left(x_{U,m} \middle| l_m, u_m \right) \tag{4.28}$$

where l_m and u_m denote the lower and upper bounds of the m-th uniform random variable $x_{U,m}$ in the vector $\mathbf{x}_U$ respectively, and $I(x_{U,m} \mid l_m , u_m)$ represents the indicator function

$$I\left(x_{U,m} \middle| l_m, u_m \right) = \begin{cases} 1 & l_m \le x_{U,m} \le u_m \\ 0 & \text{else} \end{cases} \quad \left(m = 1, 2, \cdots, M_U \right). \tag{4.29}$$

From (4.26)-(4.29), we can see that $f(\mathbf{x})$ is non-zero if and only if **x** belongs to the following set Ψ

$$\Psi = \left\{ \mathbf{x} \middle| \begin{matrix} x_{G,i} \in (-\infty, +\infty) & i = 1, 2, \cdots, M_G \\ x_{U,j} \in \left[l_j, u_j \right] & j = 1, 2, \cdots, M_U \end{matrix} \right\}. \tag{4.30}$$

Hence, the random samples drawn from $f(\mathbf{x})$ must belong to Ψ.

Similar to Section 4.1, we need to answer the following two fundamental questions when applying SSS to Gaussian-Uniform distribution:

1) How to efficiently draw random samples from the high-probability failure region given the PDF $f(\mathbf{x})$ defined in (4.26)?
2) How to estimate the failure rate based on these random samples?

The answers to these questions will be explained in the following sub-sections.

4.2.1 Statistical Sampling

In this section, we focus on the failure region Ω that sits inside the set Ψ where Ψ is defined by (4.30). For any location $\mathbf{x}$ outside the set Ψ, the probability of reaching $\mathbf{x}$ is zero and, hence, it does not contribute to the failure rate of interest. Due to this reason, we consider the failure region Ω as a subset of Ψ. For any random sample $\mathbf{x} = [\mathbf{x}_G; \mathbf{x}_U]$ falling into the failure region Ω, the uniform PDF $f_u(\mathbf{x}_U)$ is constant while the Gaussian PDF $f_g(\mathbf{x}_G)$ can be extremely small. In other words, the failure event occurs at the tail of $f_g(\mathbf{x}_G)$. To efficiently draw random samples from the high-probability failure region, we apply the idea of SSS to the Gaussian random variable $\mathbf{x}_G$. Namely, we scale up the standard deviation of $f_g(\mathbf{x}_G)$ by a factor of s ($s > 1$), while keeping $f_u(\mathbf{x}_U)$ unchanged. It, in turn, results in the following scaled PDF

$$g(\mathbf{x}) = g_g(\mathbf{x}_G) \cdot f_u(\mathbf{x}_U) \tag{4.31}$$

$$g_g(\mathbf{x}_G) = \frac{1}{\left(\sqrt{2\pi}\right)^{M_G} \cdot s^{M_G}} \exp\left(-\|\mathbf{x}_G\|_2^2 / 2s^2\right) \tag{4.32}$$

where $f_u(\mathbf{x}_U)$ is defined in (4.28). By sampling the scaled PDF $g(\mathbf{x})$ in (4.31), we can now reach the failure region Ω easily and a large number of random samples should sit inside Ω.

Since we assume that all random variables in $\mathbf{x}$ are mutually independent, we can draw random samples from the Gaussian and uniform distributions separately and then combine them together to form the random samples for the scaled PDF $g(\mathbf{x})$ in (4.31). In what follows, we will further discuss how to accurately estimate P_f in (2.15) based on these scaled random samples.

4.2.2 Failure Rate Estimation

To derive the analytical model for failure rate estimation of Gaussian-Uniform distribution, we follow the same idea presented in Section 4.1.2. Namely, we partition the M-dimensional variation space into a large number of small hyper-rectangles and the scaled failure rate P_g in (4.10) is approximated as (4.12). Substituting (4.31) into (4.12), we have

$$P_g \approx \sum_{k\in\Omega} g\left[\mathbf{x}^{(k)}\right]\cdot\Delta\mathbf{x} = \sum_{k\in\Omega} g_g\left[\mathbf{x}_G^{(k)}\right]\cdot f_u\left[\mathbf{x}_U^{(k)}\right]\cdot\Delta\mathbf{x} \tag{4.33}$$

where $\mathbf{x}^{(k)} = [\mathbf{x}_G^{(k)}; \mathbf{x}_U^{(k)}]$ denotes the center of the k-th hyper-rectangle. Substituting (4.28) and (4.32) into (4.33) yields

$$P_g \approx \sum_{k\in\Omega} \frac{\exp\left[-\left\|\mathbf{x}_G^{(k)}\right\|_2^2 \Big/ 2s^2\right]\cdot\prod_{m=1}^{M_U} I\left(x_{U,m}^{(k)}\middle| l_m, u_m\right)}{\left(\sqrt{2\pi}\right)^{M_G}\cdot s^{M_G}\cdot\prod_{m=1}^{M_U}\left(u_m - l_m\right)}\cdot\Delta\mathbf{x} \tag{4.34}$$

where $x_{U,m}^{(k)}$ represents the m-th uniform random variable in $\mathbf{x}_U^{(k)} = [x_{U,1}^{(k)}\ x_{U,2}^{(k)}\ \cdots\ x_{U,MU}^{(k)}]^T$.

Since the failure region Ω is inside the set Ψ, the indicator functions $\{I(x_{U,m}^{(k)} \mid l_m , u_m);\ m = 1, 2, \cdots, M_U,\ k \in \Omega\}$ in (4.34) are all equal to 1. Therefore, Eq. (4.34) can be further simplified as

$$P_g \approx \frac{\Delta\mathbf{x}}{\left(\sqrt{2\pi}\right)^{M_G}\cdot s^{M_G}\cdot\prod_{m=1}^{M_U}\left(u_m - l_m\right)}\cdot\sum_{k\in\Omega}\exp\left[-\left\|\mathbf{x}_G^{(k)}\right\|_2^2 \Big/ 2s^2\right]. \tag{4.35}$$

By following the mathematical analysis described in Section 4.1.2, we can approximate (4.35) as

$$\log P_g \approx \alpha + \beta\cdot\log s + \frac{\gamma}{s^2} \tag{4.36}$$

where

$$\alpha = \log\left\{\Delta\mathbf{x}\Big/\left[\left(\sqrt{2\pi}\right)^{M_G}\cdot\prod_{m=1}^{M_U}\left(u_m - l_m\right)\right]\right\} + \log\left(T_0\right) \tag{4.37}$$

$$\beta = -M_G \tag{4.38}$$

$$\gamma = -\frac{1}{2}\cdot\min_{k\in\Omega}\left[\left\|\mathbf{x}_G^{(k)}\right\|_2^2\right]. \tag{4.39}$$

In (4.37), T_0 denotes the number of dominant hyper-rectangles in the set $\{\mathbf{x}_G^{(k)};\ k \in \Omega\}$.

The analytical model in (4.36) for Gaussian-Uniform distribution is identical to that in (4.19) for

Gaussian distribution. Similar to the Gaussian distribution case, we first estimate the scaled failure rates $\{P_{g,q}; q = 1, 2, \cdots, Q\}$ by setting the scaling factor s to a number of different values $\{s_q; q = 1, 2, \cdots, Q\}$. Next, the model coefficients α, β and γ are fitted by linear regression based on $\{(s_q, P_{g,q}); q = 1, 2, \cdots, Q\}$. Once α, β and γ are known, the failure rate P_f in (2.15) can be predicted by using (4.24).

In summary, if both Gaussian and uniform distributions are used to model process variations, we only scale up the standard deviation of the Gaussian distribution without changing the uniform distribution. The failure rate P_f in (2.15) is then estimated by fitting an analytical model. The model fitting scheme is identical to what is described for Gaussian distribution in Section 4.1.

Finally, it is important to mention that the SSS method presented in this section can only handle a special class of non-Gaussian distribution where all random variables in $\mathbf{x}$ are mutually independent, a subset of these random variables are Gaussian random variables and the other random variables are uniformly distributed. How to extend SSS to other non-Gaussian distributions remains an open question and will be further studied in our future research.

To make the proposed SSS method of practical utility, a number of efficient algorithms are further studied in Section 4.3, including: (i) model fitting via maximum-likelihood estimation, (ii) confidence interval estimation via bootstrap [70], and (iii) scaling factor selection. Since the aforementioned algorithms can be generally applied to both Gaussian and Gaussian-Uniform distributions, we will not explicitly distinguish these two different cases when discussing these implementation details.

4.3 Implementation Details

4.3.1 Model Fitting via Maximum-Likelihood Estimation (MLE)

While the basic idea of SSS has been illustrated in Section 4.1-4.2, we will develop a statistically optimal algorithm to implement it in this section. Our goal is to determine the maximum-likelihood estimation (MLE) for the model coefficients α, β and γ in (4.19) and (4.36). The MLE solution can be solved from an optimization problem and it is considered to be statistically optimal for a given set of random samples.

Without loss of generality, we assume that N_q scaled random samples $\{\mathbf{x}^{(n)}; n = 1, 2, \cdots, N_q\}$ are collected for the scaling factor s_q. The scaled failure rate $P_{g,q}$ can be estimated by the brute-force Monte Carlo approach

$$P_{g,q}^{MC} = \frac{1}{N_q} \cdot \sum_{n=1}^{N_q} I\left[\mathbf{x}^{(n)}\right] \tag{4.40}$$

where $I(\mathbf{x})$ is the indicator function defined in (2.16). The variance of the estimator $P_{g,q}^{MC}$ in (4.40) can be approximated as [71]

$$v_{g,q}^{MC} = \frac{P_{g,q}^{MC} \cdot \left(1 - P_{g,q}^{MC}\right)}{N_q}. \tag{4.41}$$

If the number of samples N_q is sufficiently large, the estimator $P_{g,q}^{MC}$ in (4.40) follows a Gaussian distribution according to the central limit theorem [71]

$$P_{g,q}^{MC} \sim N\left(P_{g,q}, v_{g,q}^{MC}\right) \tag{4.42}$$

where $P_{g,q}$ denotes the actual failure rate corresponding to the scaling factor s_q.

Note that the model template in (4.19) and (4.36) are both expressed for $\log P_g$, instead of P_g. To further derive the probability distribution for $\log P_{g,q}^{MC}$, we approximate the nonlinear function log(•) by the first-order Taylor expansion around the mean value $P_{g,q}$ of the random variable $P_{g,q}^{MC}$

$$\log P_{g,q}^{MC} \approx \log P_{g,q} + \frac{P_{g,q}^{MC} - P_{g,q}}{P_{g,q}} \approx \log P_{g,q} + \frac{P_{g,q}^{MC} - P_{g,q}}{P_{g,q}^{MC}}. \tag{4.43}$$

Based on the linear approximation in (4.43), $\log P_{g,q}^{MC}$ follows the Normal distribution

$$\log P_{g,q}^{MC} \sim N\left[\log P_{g,q}, \frac{v_{g,q}^{MC}}{\left(P_{g,q}^{MC}\right)^2}\right]. \tag{4.44}$$

Eq. (4.44) is valid for all scaling factors $\{s_q;\ q = 1, 2, \cdots, Q\}$. In addition, since the scaled failure rates corresponding to different scaling factors are estimated by independent Monte Carlo simulations, the estimated failure rates $\{P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$ are mutually independent. Therefore, the Q-dimensional random variable

$$\log \mathbf{P}_g^{MC} = \left[\log P_{g,1}^{MC} \quad \log P_{g,2}^{MC} \quad \cdots \quad \log P_{g,Q}^{MC}\right]^T \tag{4.45}$$

satisfies the following jointly Normal distribution

$$\log \mathbf{P}_g^{MC} \sim N\left(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g\right) \tag{4.46}$$

where the mean vector $\boldsymbol{\mu}_g$ and the covariance matrix $\boldsymbol{\Sigma}_g$ are equal to

$$\boldsymbol{\mu}_g = \left[\log P_{g,1} \quad \log P_{g,2} \quad \cdots \quad \log P_{g,Q}\right]^T \tag{4.47}$$

$$\boldsymbol{\Sigma}_g = \begin{bmatrix} \dfrac{v_{g,1}^{MC}}{\left(P_{g,1}^{MC}\right)^2} & & & \\ & \dfrac{v_{g,2}^{MC}}{\left(P_{g,2}^{MC}\right)^2} & & \\ & & \ddots & \\ & & & \dfrac{v_{g,Q}^{MC}}{\left(P_{g,Q}^{MC}\right)^2} \end{bmatrix}. \tag{4.48}$$

The diagonal elements of the covariance matrix $\boldsymbol{\Sigma}_g$ in (4.48) can be substantially different. In other words, the accuracy of $\{\log P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$ associated with different scaling factors $\{s_q;\ q = 1, 2, \cdots, Q\}$ can be different, because the scaled failure rates $\{P_{g,q};\ q = 1, 2, \cdots, Q\}$ strongly depend on the scaling factors. In general, we can expect that if the scaling factor s_q is small, the scaled failure rate $P_{g,q}$ is small and, hence, it is difficult to accurately estimate $\log P_{g,q}$ from a small number of random samples. For this reason, instead of equally "trusting" the estimators $\{\log P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$, we must carefully model the "confidence" for each estimator $\log P_{g,q}^{MC}$, as encoded by the covariance matrix $\boldsymbol{\Sigma}_g$ in (4.48). Such "confidence" information will be fully exploited by the MLE framework to fit a statistically optimal model.

Since the scaled failure rates $\{P_{g,q};\ q = 1, 2, \cdots, Q\}$ follow the analytical model in (4.19) and (4.36), the mean vector $\boldsymbol{\mu}_g$ in (4.47) can be re-written as

$$\boldsymbol{\mu}_g = \alpha + \beta \cdot \begin{bmatrix} \log s_1 \\ \log s_2 \\ \vdots \\ \log s_Q \end{bmatrix} + \gamma \cdot \begin{bmatrix} s_1^{-2} \\ s_2^{-2} \\ \vdots \\ s_Q^{-2} \end{bmatrix} = \mathbf{A} \cdot \boldsymbol{\theta} \tag{4.49}$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & \log s_1 & s_1^{-2} \\ 1 & \log s_2 & s_2^{-2} \\ \vdots & \vdots & \vdots \\ 1 & \log s_Q & s_Q^{-2} \end{bmatrix} \tag{4.50}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}^T . \tag{4.51}$$

Eq. (4.49) implies that the mean value of the Q-dimensional random variable $\log\mathbf{P}_g^{MC}$ depends on the model coefficients α, β and γ. Given $\{P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$, the key idea of MLE is to find the optimal values of α, β and γ so that the likelihood of observing $\{P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$ is maximized.

Because the random variable $\log\mathbf{P}_g^{MC}$ follows the jointly Normal distribution in (4.46), the likelihood associated with the estimated failure rates $\{P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$ is proportional to

$$L \sim \exp\left[-\frac{1}{2}\left(\log\mathbf{P}_g^{MC} - \boldsymbol{\mu}_g\right)^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \left(\log\mathbf{P}_g^{MC} - \boldsymbol{\mu}_g\right) \right]. \tag{4.52}$$

Taking the logarithm for (4.52) yields

$$\log L \sim -\left(\log\mathbf{P}_g^{MC} - \boldsymbol{\mu}_g\right)^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \left(\log\mathbf{P}_g^{MC} - \boldsymbol{\mu}_g\right). \tag{4.53}$$

Substitute (4.49) into (4.53), and we have

$$\log L \sim -\left(\log\mathbf{P}_g^{MC} - \mathbf{A}\cdot\boldsymbol{\theta}\right)^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \left(\log\mathbf{P}_g^{MC} - \mathbf{A}\cdot\boldsymbol{\theta}\right). \tag{4.54}$$

Note that the log-likelihood $\log L$ in (4.54) depends on the model coefficients α, β and γ, because the vector $\boldsymbol{\theta}$ is composed of these coefficients as shown in (4.51). Therefore, the MLE solution of α, β and γ can be determined by maximizing the log-likelihood function

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad -\left(\log\mathbf{P}_g^{MC} - \mathbf{A}\cdot\boldsymbol{\theta}\right)^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \left(\log\mathbf{P}_g^{MC} - \mathbf{A}\cdot\boldsymbol{\theta}\right). \tag{4.55}$$

Since the covariance matrix $\boldsymbol{\Sigma}_g$ is positive definite, the optimization in (4.55) is convex. In addition, since the log-likelihood $\log L$ is simply a quadratic function of $\boldsymbol{\theta}$, the unconstrained optimization in (4.55) can be directly solved by inspecting the first-order optimality condition [73]:

$$\frac{\partial}{\partial\boldsymbol{\theta}}\left[-\left(\log\mathbf{P}_g^{MC} - \mathbf{A}\cdot\boldsymbol{\theta}\right)^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \left(\log\mathbf{P}_g^{MC} - \mathbf{A}\cdot\boldsymbol{\theta}\right) \right] = 2\cdot\mathbf{A}^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\left(\log\mathbf{P}_g^{MC} - \mathbf{A}\cdot\boldsymbol{\theta}\right) = \mathbf{0} . \tag{4.56}$$

Based on the linear equation in (4.56), the optimal value of $\boldsymbol{\theta}$ can be determined by

$$\boldsymbol{\theta} = \left(\mathbf{A}^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\mathbf{A}\right)^{-1}\cdot\mathbf{A}^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\log\mathbf{P}_g^{MC} . \tag{4.57}$$

Studying (4.57) reveals an important fact that the estimators $\{\log P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$ are weighted by the inverse of the covariance matrix $\boldsymbol{\Sigma}_g$. Namely, if the variance of the estimator $\log P_{g,q}^{MC}$ is large, $\log P_{g,q}^{MC}$ becomes non-critical when determining the optimal values of α, β and γ. In other words, the

MLE framework has optimally weighted the importance of $\{\log P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$ based on the "confidence" level of these estimators. Once α, β and γ are solved by MLE, the original failure rate P_f can be estimated by (4.24).

4.3.2 Confidence Interval Estimation

While the MLE algorithm presented in the previous sub-section is able to optimally estimate the model coefficients α, β and γ and then predict the failure rate P_f, it remains an open question how we can quantitatively assess the accuracy of our SSS method. Since SSS is based upon Monte Carlo simulation, a natural way for accuracy assessment is to calculate the confidence interval of the estimator P_f^{SSS}. However, unlike the traditional estimator where a statistical metric is estimated by the average of multiple random samples and, hence, the confidence interval can be derived as a closed-form expression, our proposed estimator P_f^{SSS} is calculated by linear regression with nonlinear exponential/logarithmic transformation, as shown in Section 4.3.1. Accurately estimating the confidence interval of P_f^{SSS} is not a trivial task.

In this thesis, we apply bootstrap [70] to address the aforementioned challenge. The key idea of bootstrap is to re-generate a large number of random samples based on a statistical model without running additional transistor-level simulations. These random samples are then used to repeatedly calculate the value of P_f^{SSS} in (4.24) for multiple times. Based on these repeated runs, the statistics (hence, the confidence interval) of the estimator P_f^{SSS} can be accurately estimated.

In particular, we start from the estimated failure rates $\{P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$. Each estimator $P_{g,q}^{MC}$ follows the Normal distribution in (4.42). The actual mean $P_{g,q}$ in (4.42) is unknown; however, we can approximate its value by the estimated failure rate $P_{g,q}^{MC}$. Once we know the statistical distribution of $P_{g,q}^{MC}$, we can re-sample its distribution and generate N_{RS} sampled values $\{P_{g,q}^{MC(n)};\ n = 1, 2, \cdots, N_{RS}\}$. This re-sampling idea is applied to all scaling factors $\{s_q;\ q = 1, 2, \cdots, Q\}$, thereby resulting in a large data set $\{P_{g,q}^{MC(n)};\ q = 1, 2, \cdots, Q,\ n = 1, 2, \cdots, N_{RS}\}$. Next, we repeatedly run SSS for N_{RS} times and get N_{RS} different failure rates $\{P_f^{SSS(n)};\ n = 1, 2, \cdots, N_{RS}\}$. The confidence interval of P_f^{SSS} can then be estimated from the statistics of these failure rate values.

4.3.3 Scaling Factor Selection

In Section 4.3.1-4.3.2, we assume that a set of pre-selected scaling factors $\{s_q; q = 1, 2, \cdots, Q\}$ are already given. In practice, appropriately choosing these scaling factors is a critical task due to several reasons. First, if these scaling factors are too large, the estimator P_f^{SSS} based on extrapolation in (4.24) would not be accurate, since the extrapolation point $s = 1$ is far away from the selected scaling factors. Second, if the scaling factors are too small, the scaled failure rates $\{P_{g,q}; q = 1, 2, \cdots, Q\}$ are extremely small and they cannot be accurately estimated from a small number of scaled random samples. Hence, finding an appropriate set of scaling factors can be extremely challenging.

In this thesis, we apply a heuristic approach to determine the scaling factors. The heuristic approach consists of two stages: (1) sample allocation and (2) scaling factor selection. The first stage (i.e., sample allocation) assumes that

1) $\log(P_g)$ is a linear function of the scaling factor s, as shown in Figure 4-2. This linear assumption makes sample allocation feasible even if we have no information about the circuits, as will be seen later.

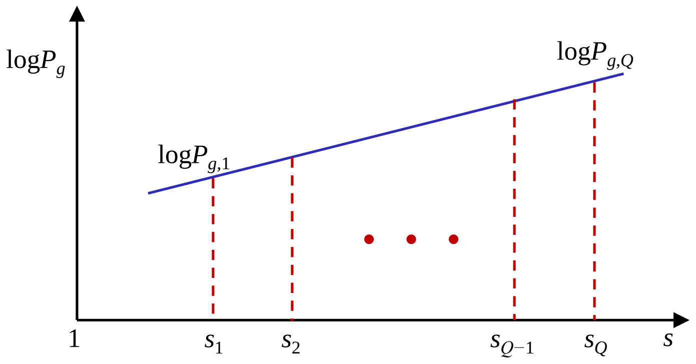

Figure 4-2. $\log(P_g)$ is assumed to be a linear function of the scaling factor s at sample allocation stage.

2) The scaling factors $\{s_q; q = 1, 2, \cdots, Q\}$ are evenly distributed.

3) The number of failure samples must be equal to or larger than a pre-defined number (e.g., 20) at each scaling factor. As mentioned in Section 4.3.1, the brute-force Monte Carlo approach is applied to estimate the scaled failure rate for each scaling factor. From (2.31), we can see that the estimation accuracy of the brute-force Monte Carlo approach is mainly determined by the number of failure samples. Therefore, this assumption guarantees the estimation accuracy at each scaling factor.

4) The scaled failure rates must be equal to or smaller than a pre-defined number (e.g., 0.3). Generally speaking, a larger scaling factor corresponds to a larger scaled failure rate. Hence, this assumption constrains the distance between the selected scaling factors and $s = 1.0$.

Given these assumptions, we allocate samples for each scaling factor by the following algorithm:

Algorithm 4.1 Sample Allocation for SSS

1. Start with the total number of samples (say, N), the number of scaling factors (say, Q), the minimum number of failure samples at each scaling factor (say, N_{min}), and the maximum scaled failure rate at selected scaling factors (say, P_{max}). Set $N_{step} = 100$.

2. Initialize $P_{g,q}^* = P_{max}$ and $N_q = N_{min} / P_{g,q}^*$, where $q = 1, 2, \cdots, Q$. Here, $P_{g,q}^*$ denotes the target scaled failure rate at the q-th scaling factor s_q and N_q denotes the number of samples that we allocate for s_q. Without losing generality, s_1 denotes the smallest scaling factor and s_Q denotes the largest scaling factor.

3. If $(N_1 + N_2 + \cdots + N_Q) < N$, go to Step 4. Otherwise, go to Step 7.

4. $N_1 = N_1 + N_{step}$. Update $P_{g,1}^*$

$$P_{g,1}^* = \frac{N_{\min}}{N_1}. \tag{4.58}$$

5. Based on the linear assumption illustrated in Figure 4-2, update $\{P_{g,q}^*; q = 2, 3, \cdots, Q-1\}$

$$P_{g,q}^* = \exp\left[\frac{q-1}{Q-1}\cdot\left(\log P_{g,Q}^* - \log P_{g,1}^*\right) + \log P_{g,1}^*\right] \quad q = 2,3,\cdots,Q-1. \tag{4.59}$$

6. Update $\{N_q; q = 2, 3, \cdots, Q-1\}$

$$N_q = \frac{N_{\min}}{P_{g,q}^*} \quad q = 2,3,\cdots,Q-1. \tag{4.60}$$

Go to Step 3.

7. Normalize $\{N_q; q = 1, 2, \cdots, Q\}$

$$N_q^* = \text{floor}\left[\left(N\cdot N_q\right)\Big/\sum_{q=1}^{Q} N_q\right] \quad q = 1,2,\cdots,Q. \tag{4.61}$$

where floor(•) maps a real number to the largest previous integer, and $\{N_q^*; q = 1, 2, \cdots, Q\}$ denote the normalized numbers.

There are several important clarifications that we need to make for Algorithm 4.1. First, the scaled failure rates $\{P_{g,q}^*; q = 1, 2, \cdots, Q\}$ are our target failure rates corresponding to the evenly distributed scaling factors $\{s_q; q = 1, 2, \cdots, Q\}$. $\{N_q^*; q = 1, 2, \cdots, Q\}$ are the number of samples that we intend to generate for $\{s_q; q = 1, 2, \cdots, Q\}$ in order to estimate $\{P_{g,q}^*; q = 1, 2, \cdots, Q\}$. Note that we do not know $\{s_q; q = 1, 2, \cdots, Q\}$ when running Algorithm 4.1. The reason that we can compute $\{P_{g,q}^*; q = 1, 2, \cdots, Q\}$ and $\{N_q^*; q = 1, 2, \cdots, Q\}$ without explicitly knowing $\{s_q; q = 1, 2, \cdots, Q\}$ is because of the linear assumption, as illustrated in Figure 4-2. Second, Algorithm 4.1 aims to select scaling factors that have small scaled failure rates under the constraints that the minimum number of failure samples at selected scaling factors must be equal to or larger than a pre-defined number N_{min} and the total number of samples must be equal to or smaller than a pre-defined number N. Alternatively speaking, Algorithm 4.1 intends to select small scaling factors without violating the constraints, which guarantee the estimation accuracy at selected scaling factors. Third, Algorithm 4.1 does not utilize any information about the circuits. The information of the circuits will be used when we determine $\{s_q; q = 1, 2, \cdots, Q\}$ in the second stage (i.e., scaling factor selection). Last, Algorithm 4.1 does not perform any transistor-level simulations. Hence, the computational cost of running Algorithm 4.1 is negligible.

Next, we need to determine $\{s_q; q = 1, 2, \cdots, Q\}$ given $\{P_{g,q}^*; q = 1, 2, \cdots, Q\}$ and $\{N_q^*; q = 1, 2, \cdots, Q\}$ at the second stage, which consists of two steps:

1) Find s_1 and s_Q.

2) Determine $\{s_q; q = 2, 3, \cdots, Q-1\}$ based on the assumption that $\{s_q; q = 1, 2, \cdots, Q\}$ are evenly distributed.

We can apply various search algorithms (e.g., binary search, etc.) to find s_1 and s_Q. For instance, we can apply binary search to find s_Q. The simplified searching procedure is described in Algorithm 4.2.

Algorithm 4.2 Binary Search for s_Q

1. Start with a pre-defined number N_{batch}, the default scaling factor lower bound s_{low}, the default scaling factor upper bound s_{up}, the scaling factor resolution s_{step} (e.g., 0.1), N_{min}, and N_Q^*. Here, N_{min} is defined in Algorithm 4.1, and N_Q^* is determined after running Algorithm 4.1.
2. If $s_{up} - s_{low} \geq 2\times s_{step}$, go to Step 3. Otherwise, go to Step 4.
3. Compute s_Q as $s_Q = (s_{low} + s_{up})/2$. Generate N_Q^*/N_{batch} samples and perform transistor-level

simulations to evaluate their performance values. Denote the number of failure samples as N_{Fail}. Proceed according to the following rules:

$$\begin{cases} \text{Set } s_{low} = s_Q & \text{if } N_{Fail} < \dfrac{N_{\min}}{N_{batch}} - \Delta \\ \text{The algorithm is complete} & \text{if } \dfrac{N_{\min}}{N_{batch}} - \Delta \le N_{Fail} \le \dfrac{N_{\min}}{N_{batch}} + \Delta \\ \text{Set } s_{up} = s_Q & \text{if } N_{Fail} > \dfrac{N_{\min}}{N_{batch}} + \Delta \end{cases} \tag{4.62}$$

where Δ is a pre-defined margin number and used to reduce the binary search cost. Go to Step 2.

4. Binary search fails to find our desired scaling factor s_Q.

There are several clarifications that we need to make for Algorithm 4.2. First, Algorithm 4.2 assumes that the scaled failure rate monotonically increases with the scaling factor s. In general, this assumption holds. However, there are some special scenarios where such assumption fails to work, which will be discussed in detail in Section 4.5. Once such assumption does not hold, binary search described in Algorithm 4.2 would fail to find our desired scaling factor s_Q. Second, the scaling factor resolution s_{step} and the margin number Δ are used to reduce the number of iterations in binary search. In other words, these two parameters facilitate efficient search of s_Q. Third, if $s_{up} - s_{low}$ is too small, we cannot find an appropriate scaling factor given the default scaling factor range. To fix this, we need to increase the default scaling factor upper bound if $N_{Fail} < N_{min}/\text{N}_{batch} - \Delta$, or decrease the default scaling factor lower bound if $N_{Fail} > N_{min}/\text{N}_{batch} + \Delta$.

Once s_1 and s_Q are determined, $\{s_q;\ q = 2, 3, \cdots, Q-1\}$ can be computed by

$$s_q = \frac{q-1}{Q-1} \cdot \left(s_Q - s_1\right) + s_1 \qquad q = 2, 3, \cdots, Q-1\,. \tag{4.63}$$

Once $\{s_q;\ q = 1, 2, \cdots, Q\}$ are available, we need to re-calculate the number of samples that we intend to generate for $\{s_q;\ q = 1, 2, \cdots, Q\}$ because a number of samples are consumed by the search algorithm at Stage 2. Let us use $\{N_q^{search};\ q = 1, 2, \cdots, Q\}$ to denote the number of samples that we generate for $\{s_q;\ q = 1, 2, \cdots, Q\}$ at Stage 2, and N^{search} to denote the total number of samples that we spend at Stage 2. There are two things that we need to notice here. First, N^{search} is equal to or larger than the summation of $\{N_q^{search};\ q = 1, 2, \cdots, Q\}$. Second, N_q^{search} could be 0, where $q \in 1, 2, \cdots, Q$. To clearly understand these, let us consider a synthetic example where Q = 5, as shown in Figure 4-3. In Figure 4-3, we use blue color to highlight the scaling factors that we test when searching s_1 and s_Q, and green color to highlight the scaling factors that we eventually select for SSS. For this synthetic example, s_1 is 1.6 and s_Q is 2.4. To determine s_1 and s_Q, the search algorithm tests s = {1.5, 1.6, 1.9, 2.1, 2.4 and 2.6}. In other words, the search algorithm generates samples at s = {1.5, 1.6, 1.9, 2.1, 2.4 and 2.6}, and N^{search} is the total number of samples that we generate at s = {1.5, 1.6, 1.9, 2.1, 2.4 and 2.6}. Based on s_1 and s_Q, we select {1.6, 1.8, 2.0, 2.2 and 2.4} for SSS. From Figure 4-3, we observe that N_2^{search}, N_3^{search} and N_4^{search} are 0 since s = {1.8, 2.0 and 2.2} are not searched.

Scaling factor	1.5	1.6	1.7	1.8	1.9	2.0	2.1	2.2	2.3	2.4	2.5	2.6
Searched	🟦	🟦			🟦		🟦			🟦		🟦
Selected		🟩		🟩		🟩		🟩		🟩		

Figure 4-3. A synthetic example is used to illustrate the relation between $\{N_q^{search};\ q = 1, 2, \cdots, Q\}$ and N^{search}.

The number of samples that we intend to generate for $\{s_q;\ q = 1, 2, \cdots, Q\}$ are re-calculated as

$$N_q = \frac{N_q^*}{N} \cdot \left(N - N^{search} + \sum_{q=1}^{Q} N_q^{search} \right) \quad q = 1, 2, \cdots, Q \,. \tag{4.64}$$

4.3.4 Algorithm Summary

Algorithm 4.3 Scaled-Sigma Sampling (SSS)

1. Given the total number of simulations N, and the number of scaling factors Q.
2. Select a set of scaling factors $\{s_q;\ q = 1, 2, \cdots, Q\}$ and determine the number of samples $\{N_q;\ q = 1, 2, \cdots, Q\}$ that we intend to generate for $\{s_q;\ q = 1, 2, \cdots, Q\}$ by using the approach mentioned in Section 4.3.3.
3. For each scaling factor s_q where $q \in \{1, 2, \cdots, Q\}$, sample the scaled PDF $g(\mathbf{x})$ in (4.1) for Gaussian distribution or (4.31) for Gaussian-Uniform distribution by setting $s = s_q$, and generate N_q scaled random samples by running transistor-level simulations.

4. Calculate the scaled failure rates $\{P_{g,q}^{MC}; q = 1, 2, \cdots, Q\}$ by (4.40) based on the simulation results collected in Step 3.

5. For each estimator $P_{g,q}^{MC}$ where $q \in \{1, 2, \cdots, Q\}$, calculate its variance $v_{g,q}^{MC}$ by (4.41).

6. Form the Q-dimensional vector $\log\mathbf{P}_g^{MC}$ by taking the logarithm for the estimated failure rates $\{P_{g,q}^{MC}; q = 1, 2, \cdots, Q\}$, as shown in (4.45).

7. Form the diagonal matrix $\mathbf{\Sigma}_g$ in (4.48) and the matrix $\mathbf{A}$ in (4.50).

8. Calculate the MLE solution $\boldsymbol{\theta}$ based on (4.57), where the vector $\boldsymbol{\theta}$ is composed of the model coefficients α, β and γ as shown in (4.51).

9. Approximate the failure rate P_f by the estimator P_f^{SSS} in (4.24).

10. For each estimator $P_{g,q}^{MC}$ where $q \in \{1, 2, \cdots, Q\}$, re-sample the Gaussian distribution in (4.42) for which the actual mean $P_{g,q}$ is approximated by its estimated value $P_{g,q}^{MC}$, and generate N_{RS} re-sampled values $\{P_{g,q}^{MC(n)}; n = 1, 2, \cdots, N_{RS}\}$.

11. For each data set $\{P_{g,q}^{MC(n)}; q = 1, 2, \cdots, Q\}$ where $n \in \{1, 2, \cdots, N_{RS}\}$, repeat Step 5~9 to calculate the failure rate $P_f^{SSS(n)}$.

12. Based on the data set $\{P_f^{SSS(n)}; n = 1, 2, \cdots, N_{RS}\}$, estimate the confidence interval of the estimator P_f^{SSS}.

There is one thing that we need to emphasize for Algorithm 4.3. While most traditional methods cannot efficiently estimate the rare failure rate in a high-dimensional variation space, the proposed SSS algorithm does not suffer from such a dimensionality problem. None of the steps in Algorithm 4.3 is sensitive to the dimensionality of the variation space. As will be demonstrated in Section 4.4, SSS achieves superior accuracy over the traditional importance sampling method when the dimensionality of the variation space exceeds a few hundred.

4.4 Experimental Results

In this section, two circuit examples are used to demonstrate the efficacy of the proposed SSS method. For testing and comparison purposes, three different approaches are implemented: (i) the brute-force Monte Carlo approach, (ii) the minimum-norm importance sampling (MNIS), and (iii) the proposed

SSS method. The brute-force Monte Carlo approach is used to generate the "golden" failure rate so that the accuracy of MNIS and SSS can be quantitatively evaluated. The implementation of MNIS consists of two stages. In the first stage, 2000 transistor-level simulations are used to search the variation space and find the failure event that is most likely to occur. Next, importance sampling is applied with a shifted Normal distribution to estimate the rare failure rate. On the other hand, when implementing the proposed SSS method, 5 different scaling factors are empirically chosen to estimate the failure rate and 200 re-sampled data points are generated to estimate the confidence interval (i.e., Q = 5 and N_{RS} = 200) in Algorithm 4.3.

4.4.1 Sense Amplifier

Shown in Figure 4-4 is the simplified circuit block diagram for a sense amplifier (SA) designed in a 45nm CMOS process. SA is composed of three blocks: (i) control signal generation block, (ii) bit line sensing block, and (iii) output driver block. The bit line sensing block is the core component of SA. Once the bit line sensing block is enabled, voltages on two bit lines (i.e., *BL* and *BLB*) are sensed and the voltage difference between *BL* and *BLB* is amplified. If V_{BL} is larger than V_{BLB}, the bit line sensing block is supposed to output 1. Otherwise, it outputs 0. In this example, the *BL* and *BLB* voltages are initially set to 1.1V and 1.2V respectively. If the output of SA is 0, the SA is considered as "PASS". Otherwise, it is considered as "FAIL".

In this SA example, we have 45 transistors and each transistor consists of a number of multipliers. To consider process variations, 4 independent Normal random variables are used to model the random mismatch of each multiplier in the process design kit. In total, there are 552 independent Normal random variables.

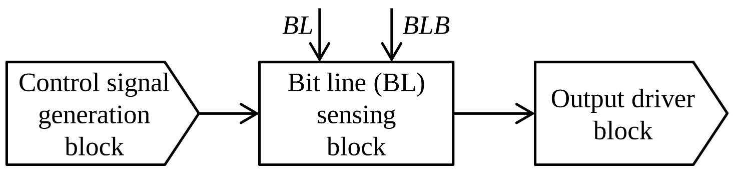

Figure 4-4. Simplified circuit block diagram is shown for a sense amplifier.

We first apply the brute-force Monte Carlo approach with 3.5×10^6 random samples and the estimated failure rate is 7.1×10^{-6}. It is considered as the "golden" result to evaluate the accuracy for other two methods in our experiment. Next, we apply the proposed SSS method (i.e., Algorithm 4.3) to estimate the failure rate. Figure 4-5 shows 5 empirically selected scaling factors $\{s_q;\ q = 1, 2, \cdots, 5\}$ and their corresponding scaled failure rates $\{P_{g,q}^{MC};\ q = 1, 2, \cdots, 5\}$. In total, 10^4 transistor-level simulations are used to search and generate these 5 data points $\{(s_q,\ P_{g,q}^{MC});\ q = 1, 2, \cdots, 5\}$. The black curve in Figure 4-5 shows the analytical model in (4.19) that is optimally fitted by MLE. The SA failure rate is then predicted by the estimator P_f^{SSS} in (4.24) based on the extrapolation at $s = 1$. Figure 4-6 further shows the histogram generated by re-sampling, as described in Algorithm 4.3. The histogram is calculated from 200 re-sampled data points, and is used to estimate the confidence interval of the estimator P_f^{SSS}. In our experiment, we notice that the computational cost of SSS is completely dominated by the transistor-level simulations required to generate the random samples. The computational time of post-processing the sampling data in Algorithm 4.3 takes less than 0.2 second and, hence, is negligible.

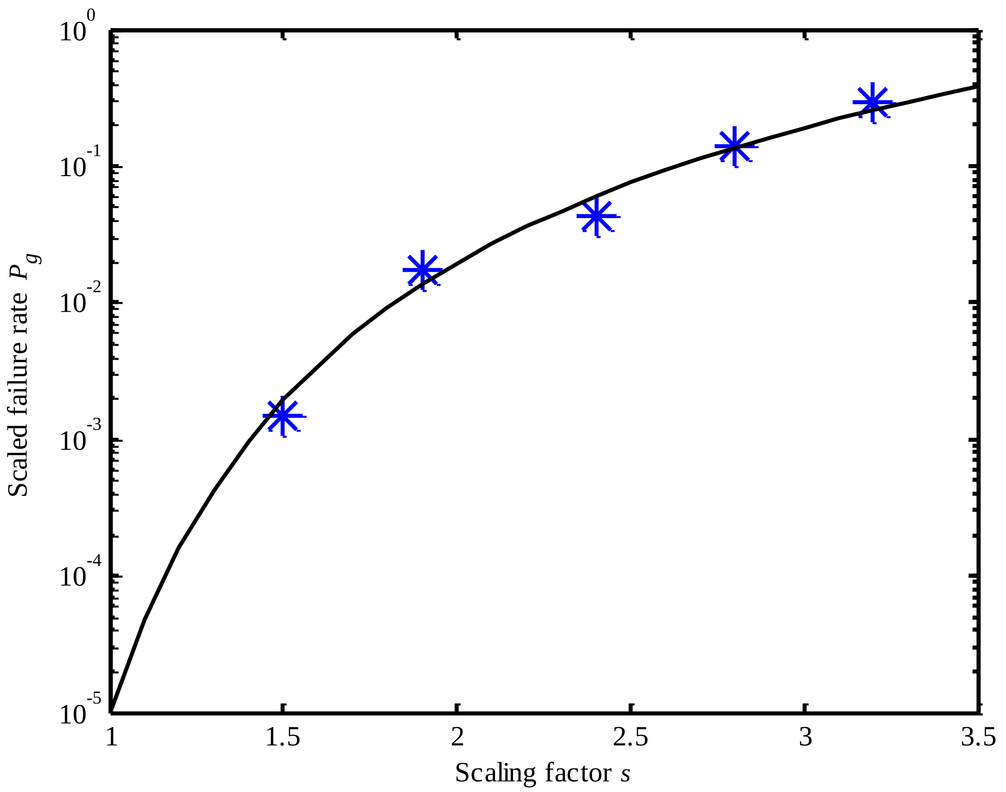

Figure 4-5. The scaled failure rate P_g is plotted as a function of the scaling factor s where the blue stars represent 5 selected scaling factors and the estimated failure rates corresponding to these scaling factors.

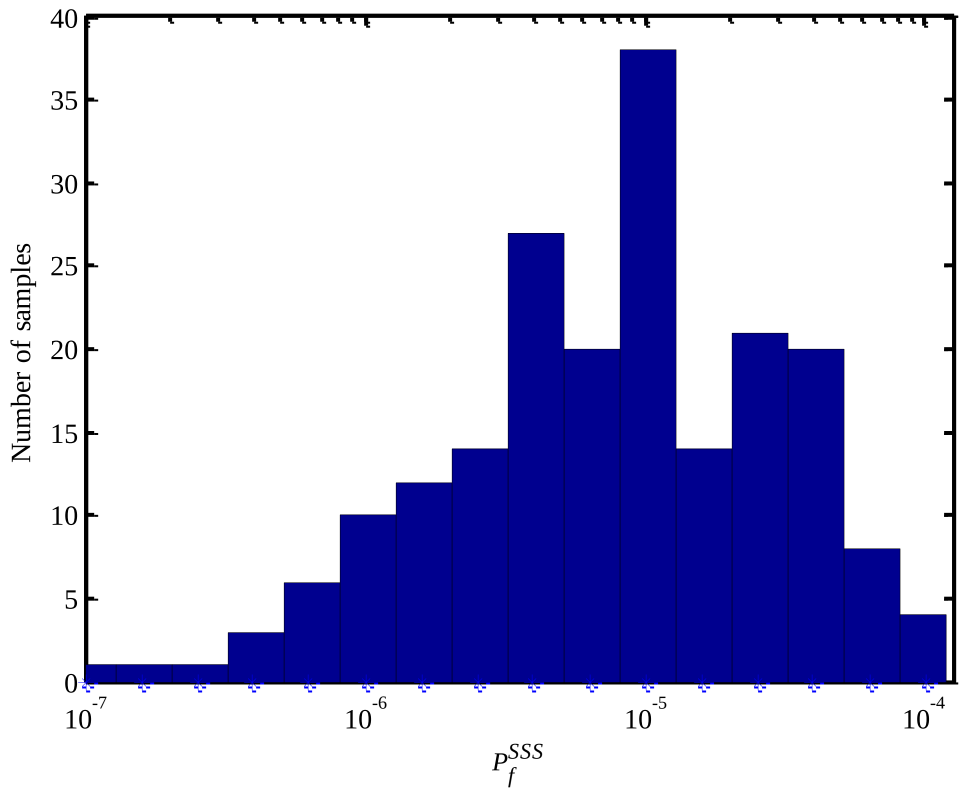

Figure 4-6. A histogram is generated by 200 re-sampled data points to estimate the confidence interval of the estimator P_f^{SSS}.

Table 4-1 compares the failure rates and the 95% confidence intervals estimated by MNIS and SSS based on different numbers of transistor-level simulations. Studying Table 4-1 reveals two important observations. First, the failure rate estimated by MNIS is substantially different from the golden result (i.e., 7.1×10^{-6}). We believe that MNIS fails to identify the critical failure region that is most likely to occur, since the variation space is high-dimensional (i.e., consisting of 552 independent random variables) in this example. Therefore, the importance sampling implemented at the second stage of MNIS fails to estimate the failure rate accurately. On the other hand, the proposed SSS method accurately estimates the failure rate.

Table 4-1. Failure rate P_f and 95% CI [P_f^L, P_f^U] estimated by MNIS and SSS ("golden" failure rate = 7.1×10^{-6})

# of Sims	MNIS	SSS

	P_f^L	P_f	P_f^U	P_f^L	P_f	P_f^U
6000	0	1.5×10^{-15}	3.1×10^{-15}	9.0×10^{-7}	4.4×10^{-5}	2.3×10^{-4}
7000	0	1.2×10^{-15}	2.5×10^{-15}	9.4×10^{-7}	3.8×10^{-5}	2.0×10^{-4}
8000	0	1.1×10^{-13}	3.1×10^{-13}	6.2×10^{-7}	2.0×10^{-5}	1.7×10^{-4}
9000	0	4.1×10^{-13}	1.1×10^{-12}	4.5×10^{-7}	1.1×10^{-5}	8.8×10^{-5}
10000	0	3.6×10^{-13}	9.2×10^{-13}	5.1×10^{-7}	1.0×10^{-5}	7.9×10^{-5}

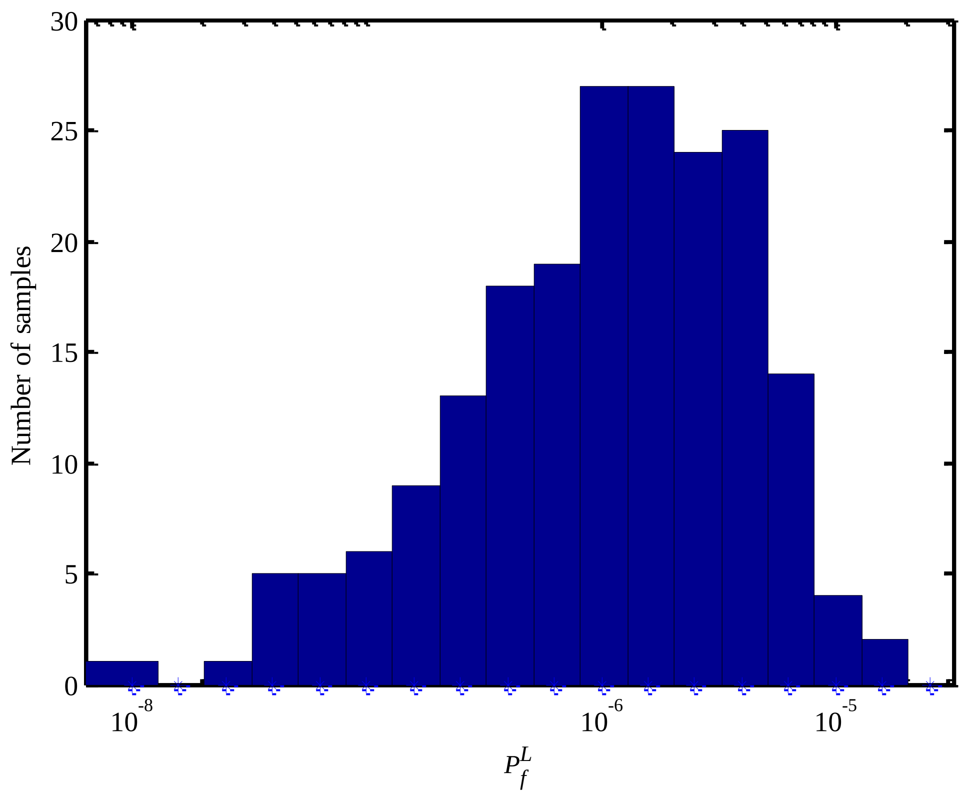

Figure 4-7. Histogram of the lower bound of the 95% confidence interval [P_f^L, P_f^U] is estimated from 200 repeated runs.

Second, but more importantly, the 95% confidence interval estimated by MNIS is not accurate either. As shown in Table 4-1, MNIS does not result in a confidence interval [P_f^L, P_f^U] that overlaps with the golden failure rate (i.e., 7.1×10^{-6}). In other words, the confidence interval estimated by MNIS based on importance sampling is highly biased. On the other hand, the confidence interval estimated by the proposed SSS method accurately covers the golden failure rate and, hence, is more reliable than the traditional MNIS approach.

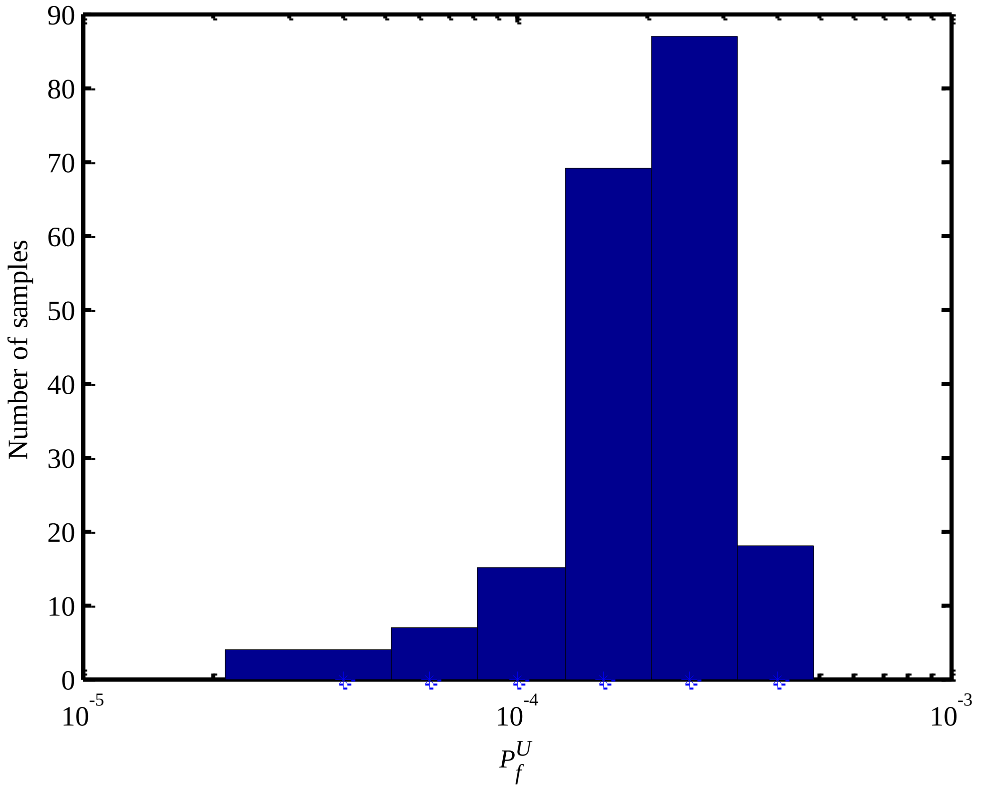

Figure 4-8. Histogram of the upper bound of the 95% confidence interval [P_f^L, P_f^U] is estimated from 200 repeated runs.

In order to further validate the confidence interval estimated by SSS, we repeatedly run Algorithm 4.3 for 200 times. During each run, the failure rate and the corresponding 95% confidence interval [P_f^L, P_f^U] are estimated from 10^4 transistor-level simulations, resulting in 200 different values for both P_f^L and P_f^U. Figure 4-7 and Figure 4-8 show the histograms of these 200 values for P_f^L and P_f^U, respectively. For only 9 cases out of 200 runs, the 95% confidence interval [P_f^L, P_f^U] does not overlap with the golden failure rate (i.e., 7.1×10^{-6}). In other words, the probability for the golden failure rate to fall out of the estimated confidence interval is 9/200 ≈ 5%. It, in turn, demonstrates that our confidence interval estimation based on bootstrap re-sampling is accurate and it is practically more attractive than the traditional MNIS method based on importance sampling.

4.4.2 SRAM Write Path

Shown in Figure 4-9 is the simplified block diagram for an SRAM write path designed in a TSMC 65nm CMOS process. The write path is composed of two components: data generation block and SRAM column block. The data generation block generates complementary values (i.e., data and datab) to drive *BL* and *BLB* to desired voltages. SRAM column block consists of 16 6-T SRAM bit-cells. In this example, we set *BL* and *BLB* to V_{DD} and 0, respectively, and then activate *WL*<0>. *BL* and *BLB* will drive bit-cell<0> from 0 to 1. To mimic the worst-case scenario, we initially store 0 in all the bit-cells, as shown in Figure 4-9. If the bit-cell<0> stores 1 after the write operation, the write path is considered as "PASS". Otherwise, the write path is considered as "FAIL". In our experiment, V_{DD} is set to 0.7V to test the circuit yield at the low power mode.

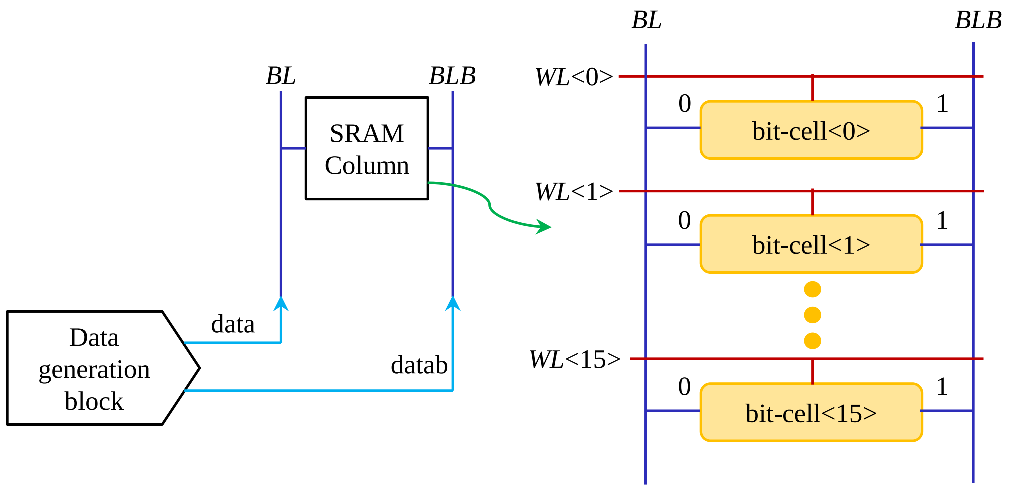

Figure 4-9. Simplified circuit block diagram is shown for an SRAM write path.

In this write path example, we have 179 transistors, and 2 independent Normal random variables are used to model the random mismatch of each transistor in the process design kit. In total, there are 358 independent Normal random variables.

We apply our proposed SSS method (i.e., Algorithm 4.3) to estimate the failure rate for this SRAM write path. Figure 4-10 shows 5 empirically selected scaling factors $\{s_q;\ q = 1, 2, \cdots, 5\}$ and their corresponding scaled failure rates $\{P_{g,q}^{MC};\ q = 1, 2, \cdots, 5\}$. In total, 10^4 transistor-level simulations are used to search and generate these 5 data points $\{(s_q,\ P_{g,q}^{MC});\ q = 1, 2, \cdots, 5\}$. The black curve in Figure 4-10 shows the analytical model in (4.19) that is optimally fitted by MLE. The write path failure rate is then predicted by the estimator P_f^{SSS} in (4.24) based on the extrapolation at $s = 1$. In this example, the estimated failure rate by SSS is 4.6×10^{-12}. The failure rate is so small that we cannot afford the brute-force Monte Carlo approach for further validation. It, in turn, demonstrates that the proposed SSS approach can efficiently estimate extremely rare failure rates with a low computational cost in a high-dimensional variation space.

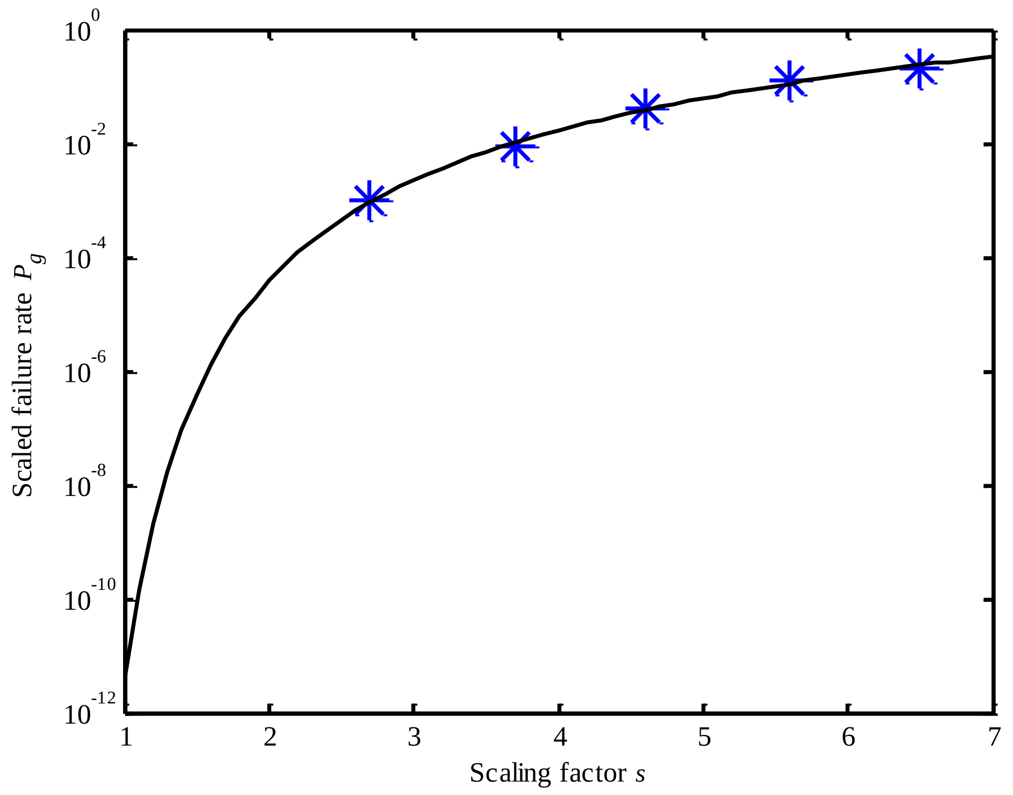

Figure 4-10. The scaled failure rate P_g is plotted as a function of the scaling factor s where the blue stars represent 5 selected scaling factors and the estimated failure rates corresponding to these scaling factors.

4.5 Algorithm Limitations

A successful application of SSS is based on several assumptions:

1. We have enough space between the smallest scaling factor s_1 and the largest scaling factor s_Q to allocate Q scaling factors given that the distance between two adjacent scaling factors cannot be smaller than the user-determined scaling factor resolution value s_{step}. In other words, the distance between s_1 and s_Q (say, $d_{S1\rightarrow SQ}$) cannot be smaller than $(Q-1)\times s_{step}$, i.e., $d_{S1\rightarrow SQ} \geq (Q-1)\times s_{step}$. In this thesis, we set Q to 5 to avoid over-fitting for the model template in (4.19) and (4.36). s_{step} is empirically set to 0.1 so that the selected scaling factors can cover a reasonable range.

   In most scenarios, this assumption holds. However, in some cases, the scaled failure rate dramatically changes with the scaling factor, making $d_{S1\rightarrow SQ} < (Q-1)\times s_{step}$. Once this occurs, SSS fails to estimate our interested failure rate. To clearly understand this scenario, let us consider synthetic test cases that have the following failure region

$$\Omega = \left\{ \mathbf{x} \,\middle|\, \sum_{m=1}^{M} x_m^2 > spec \right\} \tag{4.65}$$

where $spec$ denotes the user-defined performance specification. For illustration purposes, a 2-dimensional example with the failure region defined in (4.65) is shown in Figure 4-11, where the red color area Ω denotes the failure region.

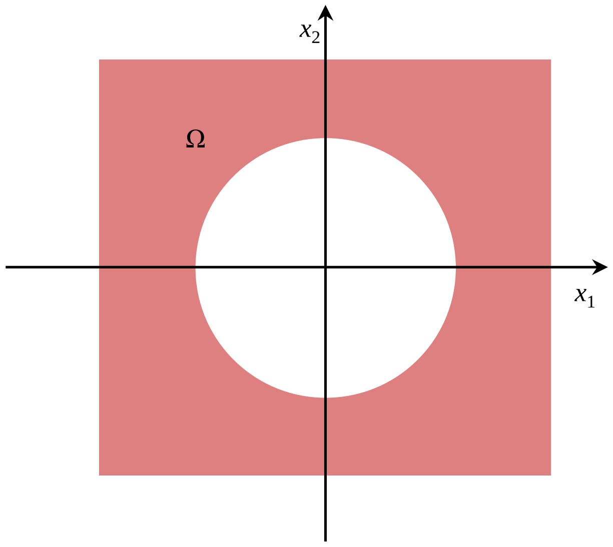

Figure 4-11. A 2-dimensional example is used to illustrate the failure region defined in (4.65).

To begin with, we set M to 10, and $spec$ to 46.9 in (4.65). The corresponding failure rate P_f is equal to 10^{-6} in this synthetic example. Then, we run SSS to estimate the failure rate. As shown in Figure 4-12, the red curve denotes the real scaled failure rate computed by the analytical calculation, and the blue curve denotes the scaled failure rate estimated by SSS. Five blue stars represent 5 scaling factors selected by SSS. The red curve and the blue curve almost overlap around s = 1.0, implying that SSS accurately estimates our interested failure rate in this synthetic example.

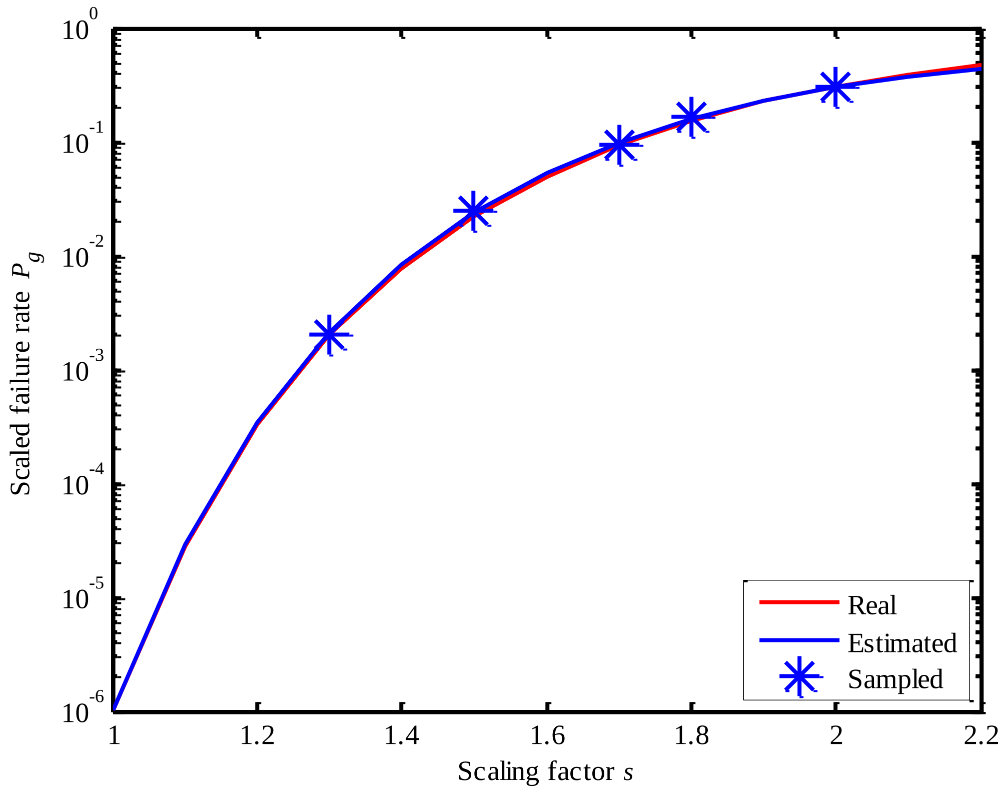

Figure 4-12. The scaled failure rate P_g is plotted as a function of the scaling factor s where the blue stars represent 5 selected scaling factors and the estimated failure rates corresponding to these scaling factors. The blue curve denotes the scaled failure rate estimated by SSS, and the red curve denotes the real scaled failure rate. The failure region is defined in (4.65) where M = 10 and $spec$ = 46.9.

Next, we set M to 100, and $spec$ to 182.1 in (4.65). The corresponding failure rate is also 10^{-6}. The scaled failure rate estimated by SSS is plotted as a function of the scaling factor, as shown in Figure 4-13. The red curve and the blue curve are very close to each other around s = 1.0 and, hence, SSS again accurately estimates the failure rate in this synthetic example.

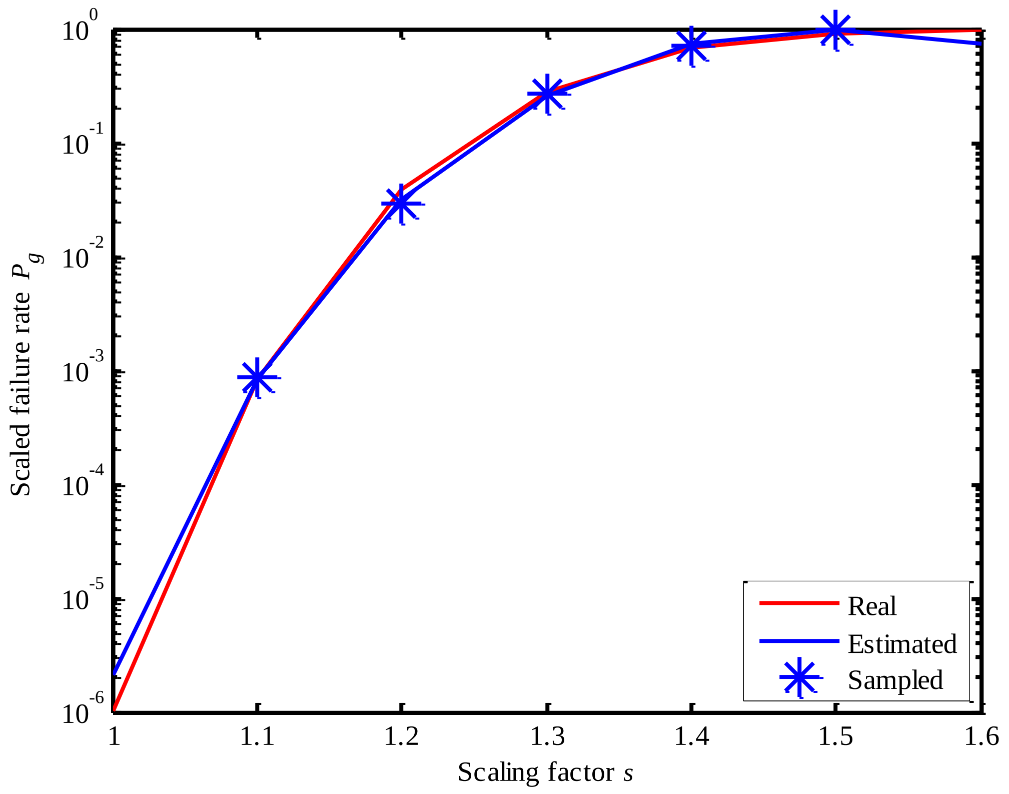

Figure 4-13. The scaled failure rate P_g is plotted as a function of the scaling factor s where the blue stars represent 5 selected scaling factors and the estimated failure rates corresponding to these scaling factors. The blue curve denotes the scaled failure rate estimated by SSS, and the red curve denotes the real scaled failure rate. The failure region is defined in (4.65) where M = 100 and *spec* = 182.1.

Studying Figure 4-12 and Figure 4-13, we observe that the scaled failure rate increases much faster at M = 100 than that at M = 10, resulting in a much smaller $d_{S1\rightarrow SQ}$ at M = 100. If we further increase M, $d_{S1\rightarrow SQ}$ will further decrease. To verify this, we increase M to 200 and set *spec* to 309.8 (the corresponding failure rate is still 10^{-6}). The largest scaling factor s_Q found by Algorithm 4.2 is 1.2 and $d_{S1\rightarrow SQ}$ = 0.1. $d_{S1\rightarrow SQ}$ is so small that we do not have enough space to allocate Q (i.e., 5) scaling factors. Because of this, SSS fails to work in this synthetic example where M = 200. Figure 4-14 shows the real scaled failure rate as a function of the scaling factor. Since SSS fails to work, we do not show any results from SSS in Figure 4-14. From Figure 4-14, we observe that the scaled failure rate dramatically increases with the scaling factor, which is the fundamental reason that the proposed SSS approach fails in this scenario.

Such failure can be detected. If SSS reports that the largest scaling factor is too close to s = 1.0, we know that the scaled failure rate dramatically increases with the scaling factor in our test case. We can apply other approaches (e.g., the brute-force Monte Carlo) to estimate the failure rate.

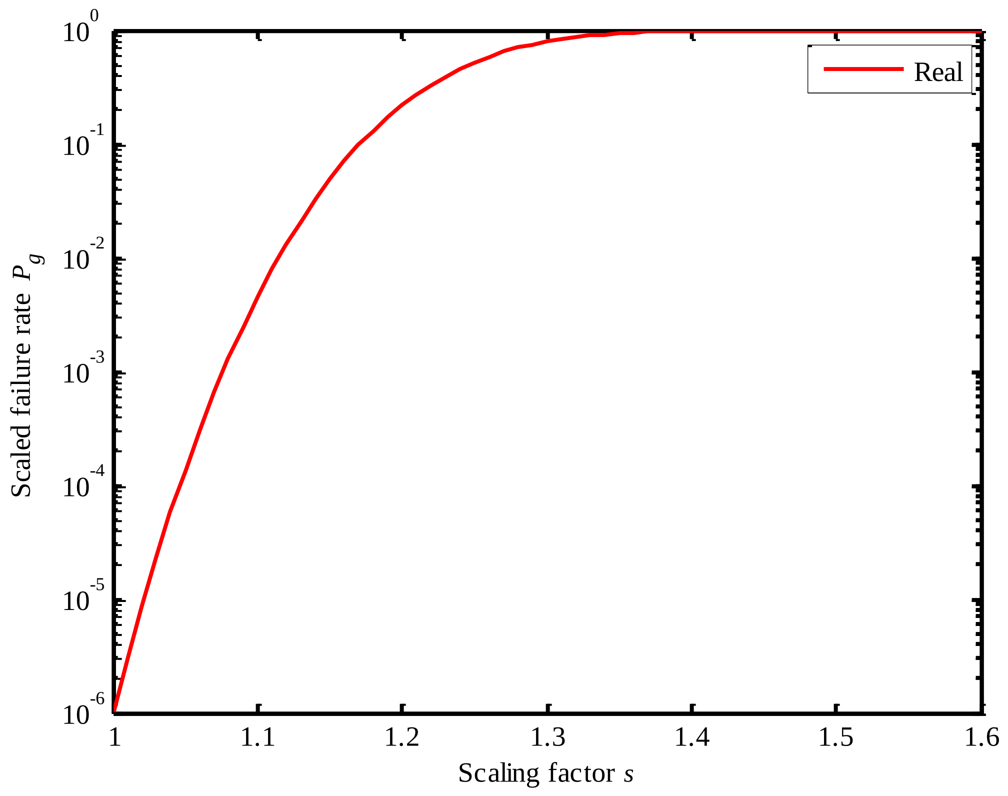

Figure 4-14. The real scaled failure rate P_g is plotted as a function of the scaling factor s. The failure region is defined in (4.65) where M = 200 and $spec$ = 309.8.

2. Another assumption that we implicitly make when applying SSS is that the scaled failure rate monotonically increases with the scaling factor. In general scenarios, this assumption is true. However, there exist some special cases where this assumption does not hold. Once this occurs, we may not get any benefit from applying SSS. More specifically, estimating scaled failure rates could be extremely expensive. In what follows, we will present two scenarios where this assumption may fail.

Scenario 1: let us consider synthetic test cases with the following failure region:

$$\Omega = \left\{ \mathbf{x} \,\middle|\, x_1 - 0.001 \cdot \sum_{m=2}^{M} x_m^2 > 4 \right\} \tag{4.66}$$

For illustration purposes, a 2-dimensional example with the failure region defined in (4.66) is shown in Figure 4-15, where the red color area Ω denotes the failure region.

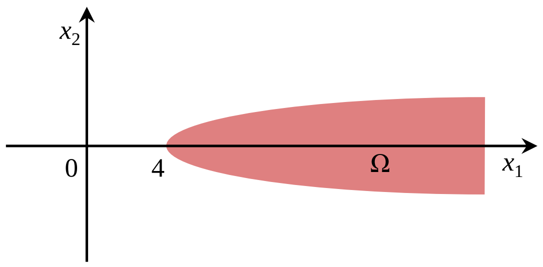

Figure 4-15. A 2-dimensional example is used to illustrate the failure region defined in (4.66).

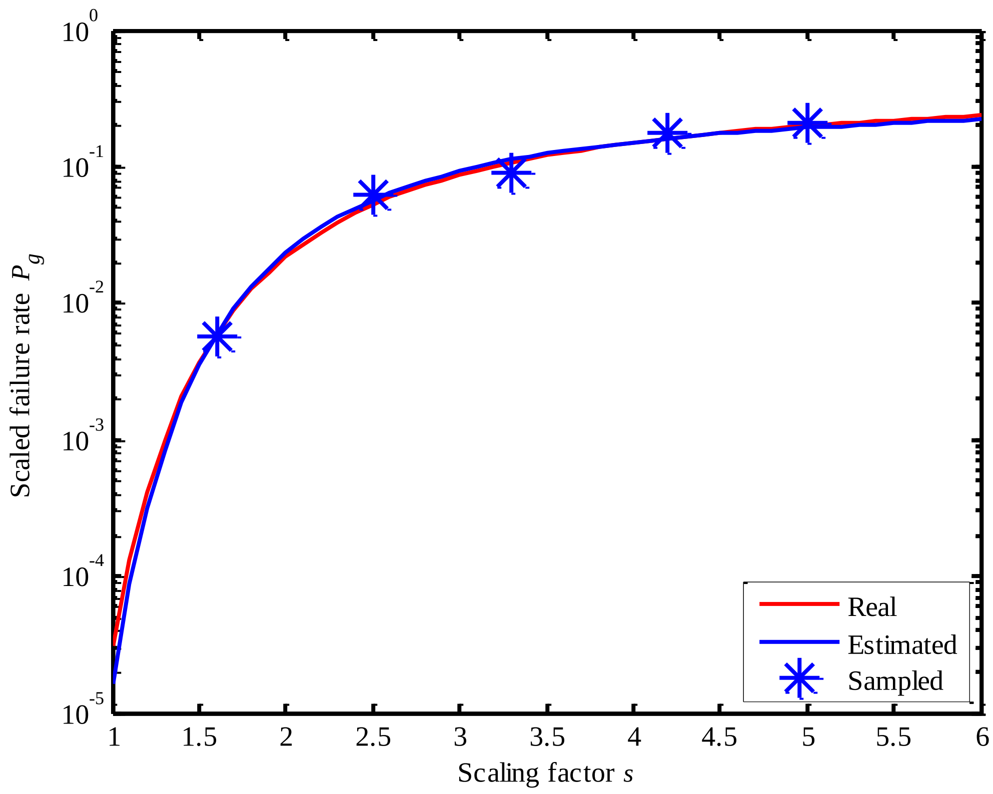

Figure 4-16. The scaled failure rate P_g is plotted as a function of the scaling factor s where the blue stars represent 5 selected scaling factors and the estimated failure rates corresponding to these scaling factors. The blue curve denotes the scaled failure rate estimated by SSS, and the red curve denotes the real scaled failure rate. The failure region is defined in (4.66) where M = 10.

We first set M to 10 in (4.66). The failure rate is 3.0×10^{-5} in this synthetic example. Then, we run SSS to estimate the failure rate. As shown in Figure 4-16, the red curve denotes the real scaled failure rate computed by the analytical calculation, and the blue curve denotes the scaled failure rate estimated by SSS. Five blue stars represent 5 scaling factors selected by SSS. The red curve and the blue curve are close to each other around s = 1.0, implying that SSS accurately estimates our interested failure rate in this synthetic example.

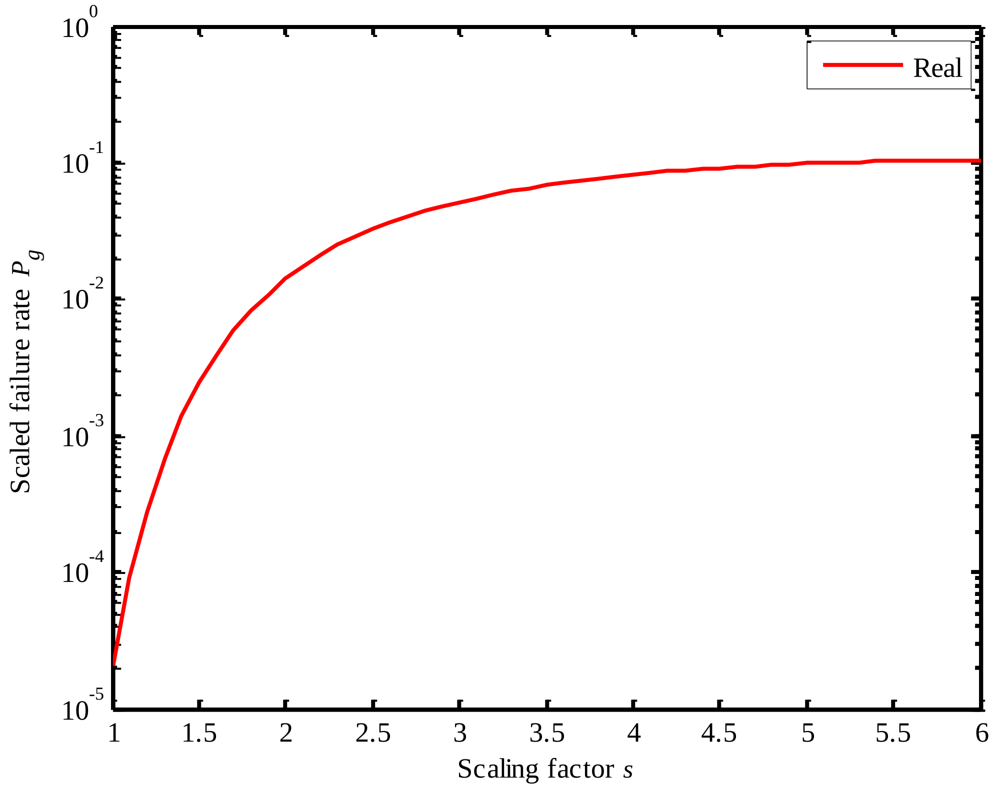

Figure 4-17. The real scaled failure rate P_g is plotted as a function of the scaling factor s. The failure region is defined in (4.66) where M = 100.

Next, we set M to 100 in (4.66). The corresponding failure rate is 2.1×10^{-5}. Figure 4-17 shows the real scaled failure rate as a function of the scaling factor. From Algorithm 4.1, we see that the scaled failure rate $P_{g,Q}$ corresponding to the largest scaling factor s_Q is equal to our pre-determined number P_{max}, which is empirically set to 0.3 in Algorithm 4.1. Algorithm 4.2 aims to find a scaling factor whose corresponding scaled failure rate is around $P_{g,Q}$, and then consider this scaling factor as s_Q.

From Figure 4-17, we can see that the scaled failure rate is much smaller than $P_{g,Q}$ even at s = 6.0. Hence, Algorithm 4.2 cannot find a scaling factor to serve as s_Q in this example. Because of this, SSS fails to work, and we do not show any results from SSS in Figure 4-17. Though an appropriate s_Q is not found in this example, the scaled failure rates corresponding to a number of scaling factors are still obtained after running Algorithm 4.2 because the scaled failure rate is easy to estimate when the scaling factor is large in this example (e.g., the scaled failure rate is 0.06 when s = 3.2). As long as we have some non-zero estimated scaled failure rates after running Algorithm 4.2, it is possible that we make SSS work for this example by decreasing the default P_{max} value in Algorithm 4.1. In other words, such failure is detectable, and could be fixed.

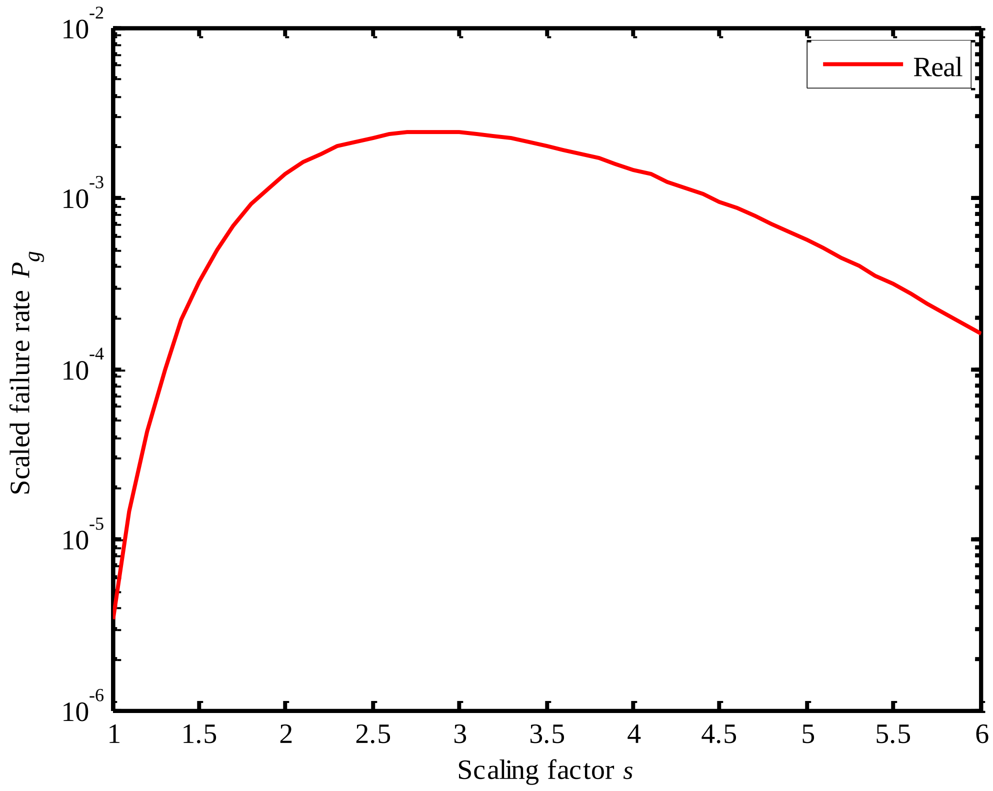

Figure 4-18. The real scaled failure rate P_g is plotted as a function of the scaling factor s. The failure region is defined in (4.66) where M = 500.

Last, we further increase M to 500. The corresponding failure rate is 3.4×10^{-6}. Figure 4-18 shows the real scaled failure rate as a function of the scaling factor. From Figure 4-18, we can see that the scaled failure rate does not monotonically increase with the scaling factor, and is not easy to estimate even if the scaling factor is very large. Similar to M = 100, we cannot find a scaling factor to serve as s_Q in this example. Because of this, SSS fails to work, and we do not show any results from SSS in Figure 4-18. Unlike M = 100 where we could get several non-zero estimated scaled failure rates after running Algorithm 4.2, all the scaled failure rates are estimated as zero since we only afford a few hundred or thousand samples in total for Algorithm 4.2. Once this occurs, we do not exactly know how to explain the result. One possible scenario is that our interested failure event is indeed very rare. Even if we increase the process variations by a very large factor, we still barely observe the failure. If this is true, we could claim that the circuit is robust. Another possible scenario is illustrated in Figure 4-18. Namely, though the scaled failure rates are not easy to estimate, our interested failure event is actually not rare. Without extra information, we do not know whether SSS fails to work or the circuit is indeed robust. In other words, such failure cannot be detected, and is hard to fix.

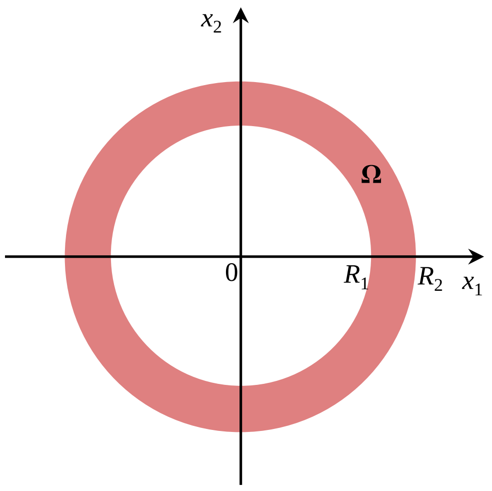

Figure 4-19. A 2-dimensional example is used to illustrate the failure region defined in (4.67).

Scenario 2: let us consider synthetic test cases with the following failure region:

$$\Omega = \left\{ \mathbf{x} \,\middle|\, R_1^2 < \sum_{m=1}^{M} x_m^2 < R_2^2 \right\} \tag{4.67}$$

where R_1 and R_2 are user-defined performance specifications. For illustration purposes, a 2-dimensional example with the failure region defined in (4.67) is shown in Figure 4-19, where the red color area Ω denotes the failure region.

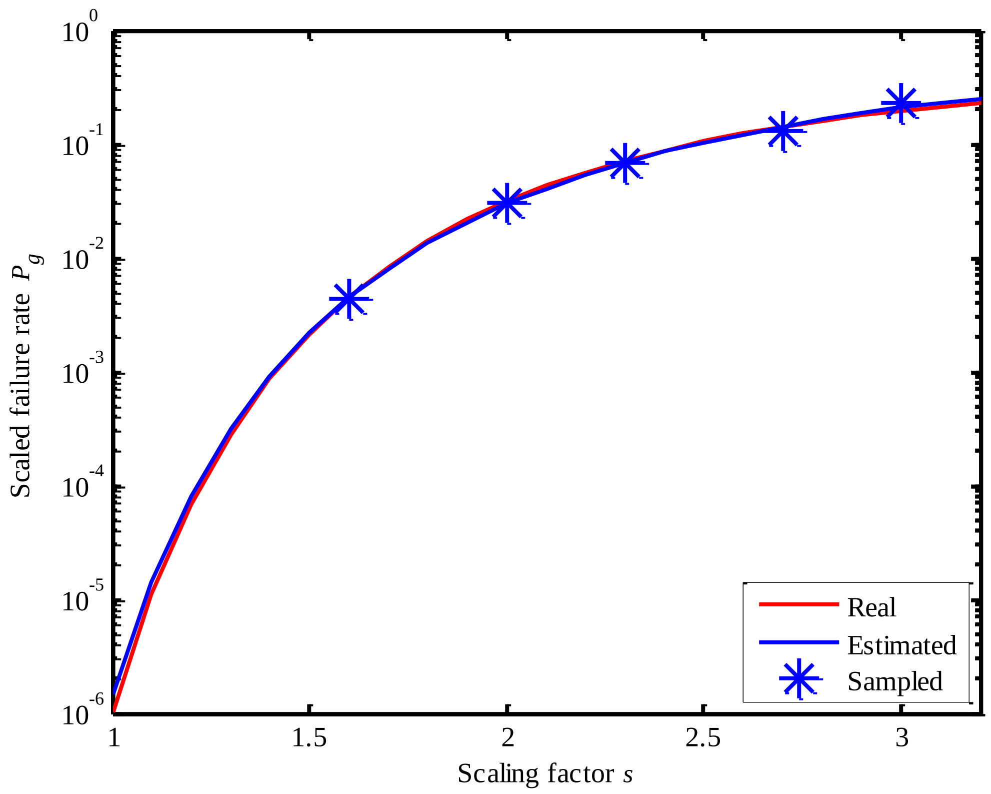

Figure 4-20. The scaled failure rate P_g is plotted as a function of the scaling factor s where the blue stars represent 5 selected scaling factors and the estimated failure rates corresponding to these scaling factors. The blue curve denotes the scaled failure rate estimated by SSS, and the red curve denotes the real scaled failure rate. The failure region is defined in (4.67) where $M = 2$, $R_1 = 5.3$ and $R_2 = 8.3$.

We first set M to 2, R_1 to 5.3 and R_2 to 8.3 in (4.67). The failure rate is 1.0×10^{-6} in this synthetic example. Then, we run SSS to estimate the failure rate. As shown in Figure 4-20, the red curve denotes the real scaled failure rate computed by the analytical calculation, and the blue curve denotes the scaled failure rate estimated by SSS. Five blue stars represent 5 scaling factors selected by SSS. The red curve and the blue curve are close to each other around $s = 1.0$, implying that SSS accurately estimates our interested failure rate in this synthetic example.

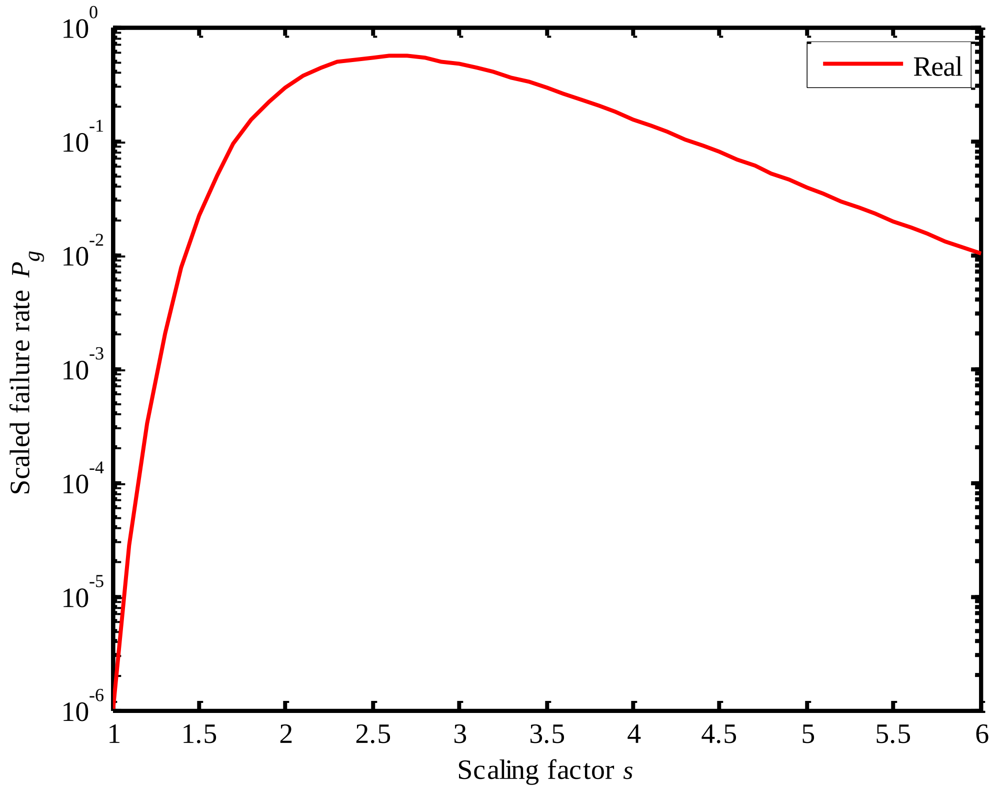

Figure 4-21. The real scaled failure rate P_g is plotted as a function of the scaling factor s. The failure region is defined in (4.67) where M = 10, R_1 = 6.8 and R_2 = 9.7.

Next, we set M to 10, R_1 to 6.8, and R_2 to 9.7 in (4.67). The corresponding failure rate is 1.0×10^{-6}. Figure 4-21 shows the real scaled failure rate as a function of the scaling factor. From Figure 4-21, we can see that the scaled failure rate does not monotonically increase with the scaling factor. Since the binary search algorithm described in Algorithm 4.2 requires monotonicity and the scaled failure rate is much smaller than P_{max} when the scaling factor is large, Algorithm 4.2 cannot find a scaling factor to serve as s_Q in this example. Because of this, SSS fails to work, and we do not show any results from SSS in Figure 4-21. Though an appropriate s_Q is not found in this example, the scaled failure rates corresponding to a number of scaling factors are still obtained after running Algorithm 4.2 because the scaled failure rate is easy to estimate when the scaling factor is not too close to s = 1.0 in this example (e.g., the scaled failure rate is 0.47 when s = 3). As long as we have some non-zero estimated scaled failure rates after running Algorithm 4.2, it is possible that we make SSS work for this example by decreasing the default P_{max} value in Algorithm 4.1. In other words, such failure is detectable, and could be fixed.

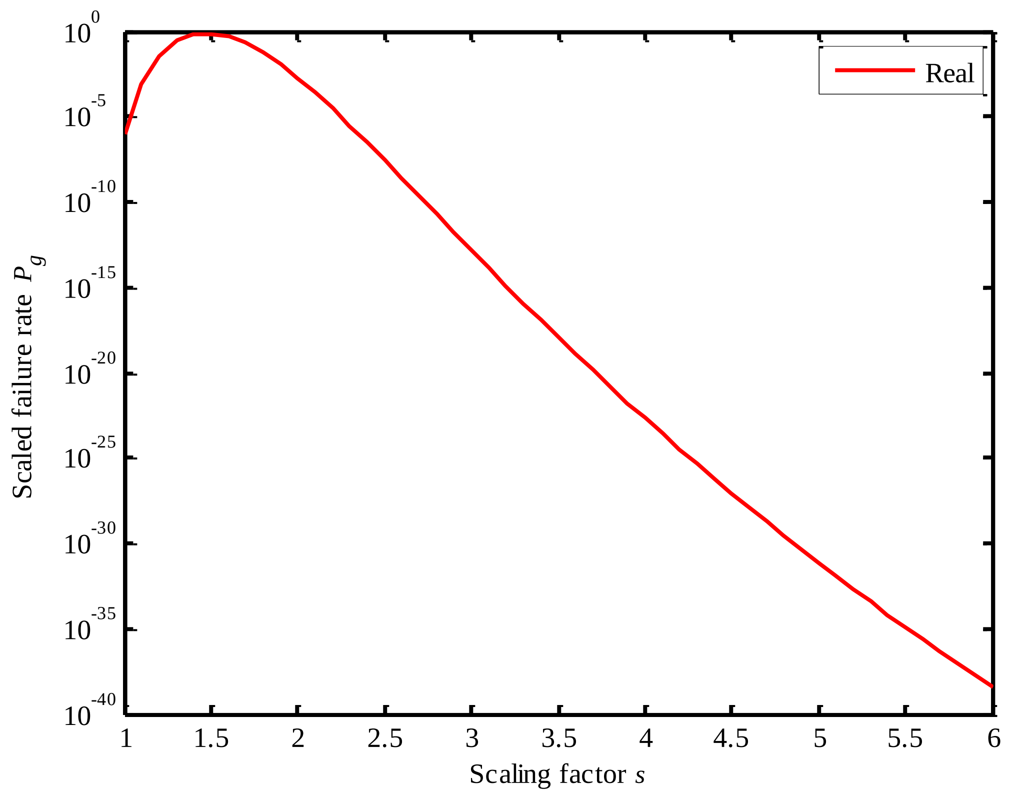

Figure 4-22. The real scaled failure rate P_g is plotted as a function of the scaling factor s. The failure region is defined in (4.67) where M = 100, R_1 = 13.5 and R_2 = 16.0.

Last, we further increase M to 100, R_1 to 13.5, and R_2 to 16.0 in (4.67). The corresponding failure rate is 1.0×10^{-6}. Figure 4-22 shows the real scaled failure rate as a function of the scaling factor. From Figure 4-22, we can see that the scaled failure rate does not monotonically increase with the scaling factor, and is almost impossible to estimate when the scaling factor is large. Similar to M = 10, we cannot find a scaling factor to serve as s_Q in this example. Because of this, SSS fails to work, and we do not show any results from SSS in Figure 4-22. Unlike M = 10 where we could get several non-zero estimated scaled failure rates after running Algorithm 4.2, all the scaled failure rates are estimated as zero since we only afford a few hundred or thousand samples in total for Algorithm 4.2. Without extra information, we do not know whether SSS fails to work or the circuit is indeed robust.

In other words, such failure cannot be detected, and is hard to fix.

For most test cases, these assumptions hold and we can safely apply the proposed SSS approach. Once these assumptions do not hold, we can apply other approaches (e.g., the brute-force Monte Carlo approach) or modify SSS (e.g., change P_{max} value in Algorithm 4.1) to fix the issue given that we can detect it. For some test cases, unfortunately, the assumptions are not satisfied and we are not able to detect it so far. One important future research direction is to develop efficient approaches to handle these cases.

Chapter 5

Bayesian Scaled-Sigma Sampling

Generally speaking, a circuit design takes several iterations to complete, as illustrated in Figure 5-1. To begin with, we start with an initial design (i.e., 1st design). If the 1st design meets all the design specifications, we are lucky and the design process is complete. Otherwise, we need to tune the 1st design to improve its performance, and we obtain a 2nd design. Next, we verify the performance of the 2nd design. If the 2nd design meets all the design specifications, we are done. Otherwise, we need to repeat the aforementioned steps until we obtain a design that can meet all the design specifications. The design process may take several iterations before we converge to the final design. In other words, we may have a number of design candidates (i.e., 1st design, 2nd design, 3rd design, 4th design, 5th design, etc.) during this process.

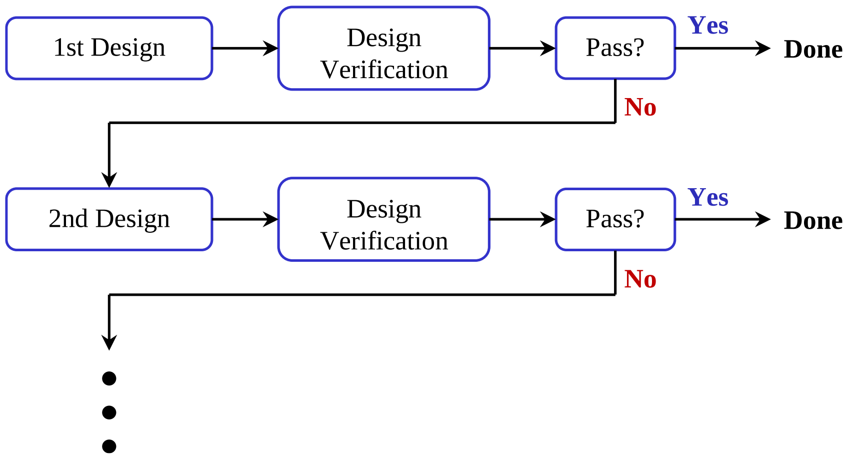

Figure 5-1. A simplified block diagram is shown for the circuit design and verification flow.

Production yield is an important design requirement and, hence, yield verification is crucial at the design verification stage. For circuits that have small failure probabilities, rare failure rate estimation approaches are applied for efficient yield verification. If the circuit has a small number of random variables, we can apply the traditional yield estimation approaches [24]-[45] to estimate the failure rate (or yield) for each design candidate. Otherwise, if the dimensionality of the circuit is large, we can apply our proposed SUS and SSS to estimate the failure rate (or yield) for each design candidate. These approaches (i.e., traditional approaches [24]-[45] and our proposed SUS and SSS) require a few thousand simulations to estimate the failure rate for each design candidate, as demonstrated in [24]-[45], Section 3.5 and Section 4.4. Since we have a number of design candidates before we converge to the final design, tens of thousands of simulations in total may be needed over the entire design process, which can be extremely expensive.

To further reduce the simulation cost, we propose a novel Bayesian scaled-sigma sampling (BSSS) approach to analyze the rare circuit failure event in the high-dimensional space. BSSS can be considered as an extension of SSS. The extension is motivated by an important observation of SSS. Studying the theoretical definition of the three model coefficients shown in (4.20)-(4.22) and (4.37)-(4.39), we observe that

- **O1**: The first two model coefficients (i.e., α and β) strongly depend on the dimensionality (i.e., M) of the variation space, but are weakly dependent of the location of the failure region Ω.
- **O2**: The third model coefficient (i.e., γ) is determined by the hyper-rectangle that is inside the failure region Ω and is closest to the origin $\mathbf{x} = \mathbf{0}$. Alternatively speaking, γ strongly depends on the location of the failure region Ω.

To fully understand the implication of this observation, let us consider two design candidates (i.e., an early design and a late design). We assume that the early design does not meet the performance specification and, hence, the late design is created from the early design by tuning its design variables (e.g., transistor sizes) to improve performance. The SSS models for these two design candidates are as follows

$$\log P_{E,g} \approx \alpha_E + \beta_E \cdot \log s + \frac{\gamma_E}{s^2} \tag{5.1}$$

$$\log P_{L,g} \approx \alpha_L + \beta_L \cdot \log s + \frac{\gamma_L}{s^2} \tag{5.2}$$

where $[\alpha_E \ \beta_E \ \gamma_E]$ and $[\alpha_L \ \beta_L \ \gamma_L]$ denote the model coefficients of the early and late designs respectively. Note that $\boldsymbol{\theta}_E = [\alpha_E \ \beta_E \ \gamma_E]^T$ is already fitted for the early design before we start to work on the late design. Our objective is to efficiently fit $\boldsymbol{\theta}_L = [\alpha_L \; \beta_L \; \gamma_L]^T$ in order to estimate the rare failure rate for the late design.

In this example, the dimensionalities of the early and late designs are likely to be similar or even identical. However, their failure regions can be different. Based on the observation (i.e., O1 and O2), we can expect that

- α_E (or β_E) and α_L (or β_L) are likely to be similar, and
- γ_E and γ_L can be substantially different.

Such a similarity between the model coefficients has not been explored by the aforementioned SSS approach. In this section, we further propose a novel Bayesian scaled-sigma sampling (BSSS) approach to take advantage of the aforementioned knowledge. In particular, BSSS encodes the "similarity" between $\boldsymbol{\theta}_E$ and $\boldsymbol{\theta}_L$ as a *prior* distribution $pdf(\boldsymbol{\theta}_L)$, and applies the Bayesian model fusion (BMF) technique [60]-[69] to solve $\boldsymbol{\theta}_L$ by maximum-a-posteriori (MAP) estimation [72]

$$\max_{\boldsymbol{\theta}_L} \quad pdf\left(\boldsymbol{\theta}_L \middle| \mathbf{D}\right) \propto pdf\left(\boldsymbol{\theta}_L\right) \cdot pdf\left(\mathbf{D} \middle| \boldsymbol{\theta}_L\right) \tag{5.3}$$

where $\mathbf{D}$ denotes the simulation data collected for the late design and $pdf(\mathbf{D} \mid \boldsymbol{\theta}_L)$ denotes the likelihood of observing the data $\mathbf{D}$.

The key difference between BSSS and SSS lies in the fact that BSSS maximizes the product of the prior distribution $pdf(\boldsymbol{\theta}_L)$ and the likelihood $pdf(\mathbf{D} \mid \boldsymbol{\theta}_L)$ by MAP, while SSS only maximizes the likelihood $pdf(\mathbf{D} \mid \boldsymbol{\theta}_L)$ by MLE. As long as the prior distribution $pdf(\boldsymbol{\theta}_L)$ is properly defined, MAP can reduce the amount of required simulation data and, hence, the model fitting cost without surrendering any accuracy, as demonstrated in [60]-[69]. In other words, BSSS can be more efficient than SSS if $pdf(\boldsymbol{\theta}_L)$ is appropriately defined.

Our proposed BSSS method consists of two major steps: (i) constructing a prior distribution $pdf(\boldsymbol{\theta}_L)$ based on the similarity between $\boldsymbol{\theta}_E$ and $\boldsymbol{\theta}_L$, and (ii) optimally determining $\boldsymbol{\theta}_L$ by MAP estimation. In what follows, we will describe these two steps in detail.

5.1 Prior Definition

Based on our knowledge of "similarity", we define α_L and β_L as two Normal random variables

$$pdf\left(\alpha_L\right)=\frac{1}{\sqrt{2\pi}\cdot\sigma_\alpha}\cdot\exp\left[-\frac{\left(\alpha_L-\alpha_E\right)^2}{2\sigma_\alpha^2}\right] \tag{5.4}$$

$$pdf\left(\beta_L\right)=\frac{1}{\sqrt{2\pi}\cdot\sigma_\beta}\cdot\exp\left[-\frac{\left(\beta_L-\beta_E\right)^2}{2\sigma_\beta^2}\right] \tag{5.5}$$

where α_E and β_E are the means of α_L and β_L respectively, and σ_α and σ_β denote the standard deviations of α_L and β_L respectively. In (5.4)-(5.5), the standard deviations σ_α and σ_β can be optimally estimated by MLE, as will be further discussed at the end of this sub-section. Because a Normal distribution peaks at its mean value, the model coefficient α_L (or β_L) for the late design is unlikely to substantially deviate from its mean value α_E (or β_E) which is the model coefficient for the early design. Restating in words, the prior distributions defined in (5.4)-(5.5) attempt to capture the similarity between α_E (or β_E) and α_L (or β_L).

On the other hand, since γ_E and γ_L can be substantially different, we have no prior knowledge about γ_L. Hence, we can only define a non-informative prior [72] for γ_L

$$pdf\left(\gamma_L\right)=\begin{cases}\dfrac{1}{u_\gamma-l_\gamma} & \gamma_L\in\left[l_\gamma,u_\gamma\right]\\ 0 & \text{else}\end{cases} \tag{5.6}$$

where l_γ and u_γ denote the lower and upper bounds for γ_L respectively. When defining $pdf(\gamma_L)$ in (5.6), we should choose l_γ to be sufficiently small and u_γ to be sufficiently large. As such, γ_L is uniformly distributed over a large range, implying that we do not know the value of γ_L in advance.

Next, we need to combine (5.4)-(5.6) to define the joint distribution for $\boldsymbol{\theta}_L = [\alpha_L\ \beta_L\ \gamma_L]^T$. Since we do not know the correlation among α_L, β_L, and γ_L, we simply assume that they are independent

$$pdf\left(\boldsymbol{\theta}_L\right)=pdf\left(\alpha_L\right)\cdot pdf\left(\beta_L\right)\cdot pdf\left(\gamma_L\right). \tag{5.7}$$

The correlation information will be learned from the data **D**, when applying MAP estimation in the next sub-section.

Last, let us discuss how to determine the values for σ_α and σ_β in (5.4)-(5.5). In this thesis, we apply MLE to determine the optimal values for σ_α and σ_β

$$\max_{\sigma_\alpha,\sigma_\beta}\quad pdf\left(\mathbf{D}\middle|\sigma_\alpha,\sigma_\beta\right) \tag{5.8}$$

where $pdf(\mathbf{D}\mid\sigma_\alpha,\sigma_\beta)$ can be expressed as

$$pdf\left(\mathbf{D}\middle|\sigma_\alpha,\sigma_\beta\right)=\int_{\boldsymbol{\theta}_L} pdf\left(\mathbf{D},\boldsymbol{\theta}_L\middle|\sigma_\alpha,\sigma_\beta\right)\cdot d\boldsymbol{\theta}_L=\int_{\boldsymbol{\theta}_L} pdf\left(\mathbf{D}\middle|\boldsymbol{\theta}_L\right)\cdot pdf\left(\boldsymbol{\theta}_L\middle|\sigma_\alpha,\sigma_\beta\right)\cdot d\boldsymbol{\theta}_L\ . \tag{5.9}$$

Substituting (5.7) into (5.9), we have

$$pdf\left(\mathbf{D}\middle|\sigma_\alpha,\sigma_\beta\right)=\int_{\boldsymbol{\theta}_L} pdf\left(\mathbf{D}\middle|\boldsymbol{\theta}_L\right)\cdot pdf\left(\alpha_L\right)\cdot pdf\left(\beta_L\right)\cdot pdf\left(\gamma_L\right)\cdot d\boldsymbol{\theta}_L\ . \tag{5.10}$$

If l_γ is sufficiently small and u_γ is sufficiently large, $pdf(\gamma_L)$ in (5.6) becomes a constant. Hence, $pdf(\mathbf{D} \mid \sigma_\alpha, \sigma_\beta)$ in (5.10) is proportional to

$$pdf\left(\mathbf{D}\middle|\sigma_\alpha,\sigma_\beta\right)\propto\int_{\boldsymbol{\theta}_L} pdf\left(\mathbf{D}\middle|\boldsymbol{\theta}_L\right)\cdot pdf\left(\alpha_L\right)\cdot pdf\left(\beta_L\right)\cdot d\boldsymbol{\theta}_L\ . \tag{5.11}$$

Given (5.4)-(5.5), $pdf(\alpha_L)\cdot pdf(\beta_L)$ in (5.11) can be expressed as

$$pdf\left(\alpha_L\right)\cdot pdf\left(\beta_L\right)=\frac{1}{2\pi\cdot\left|\boldsymbol{\Sigma}_0\right|^{1/2}}\cdot\exp\left[-\frac{1}{2}\cdot\left(\mathbf{C}\cdot\boldsymbol{\theta}_L-\boldsymbol{\theta}_0\right)^T\cdot\boldsymbol{\Sigma}_0^{-1}\cdot\left(\mathbf{C}\cdot\boldsymbol{\theta}_L-\boldsymbol{\theta}_0\right)\right] \tag{5.12}$$

where

$$\boldsymbol{\theta}_0=\begin{bmatrix}\alpha_E & \beta_E\end{bmatrix}^T \tag{5.13}$$

$$\boldsymbol{\Sigma}_0=\begin{bmatrix}\sigma_\alpha^2 & 0\\ 0 & \sigma_\beta^2\end{bmatrix} \tag{5.14}$$

$$\mathbf{C}=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}. \tag{5.15}$$

Without losing generality, we assume that Q scaling factors $\{s_q;\ q = 1, 2, \cdots, Q\}$ are selected in Algorithm 4.3, and the estimated scaled failure rates corresponding to these scaling factors are $\{P_{g,q}^{MC};\ q = 1, 2, \cdots, Q\}$. Therefore, the dataset $\mathbf{D} = \{(s_q, \log P_{g,q}^{MC});\ q = 1, 2, \cdots, Q\}$. According to (4.44), $pdf(\mathbf{D} \mid \boldsymbol{\theta}_L)$ in (5.11) can be expressed as

$$pdf\left(\mathbf{D}\middle|\boldsymbol{\theta}_L\right)=\frac{1}{\left(2\pi\right)^{Q/2}\cdot\left|\boldsymbol{\Sigma}_g\right|^{1/2}}\cdot\exp\left[-\frac{1}{2}\cdot\left(\log\mathbf{P}_g^{MC}-\mathbf{A}\cdot\boldsymbol{\theta}_L\right)^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\left(\log\mathbf{P}_g^{MC}-\mathbf{A}\cdot\boldsymbol{\theta}_L\right)\right] \tag{5.16}$$

where $\log\mathbf{P}_g^{MC}$, $\boldsymbol{\Sigma}_g$, and $\mathbf{A}$ are defined in (4.45), (4.48), and (4.50) respectively. Substituting (5.12) and (5.16) into (5.11), we have

$$pdf\left(\mathbf{D}\middle|\sigma_\alpha,\sigma_\beta\right)\propto\left|\boldsymbol{\Sigma}_0\right|^{-1/2}\cdot\int_{\boldsymbol{\theta}_L}\exp\begin{bmatrix}-\frac{1}{2}\cdot\left(\log\mathbf{P}_g^{MC}-\mathbf{A}\cdot\boldsymbol{\theta}_L\right)^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\left(\log\mathbf{P}_g^{MC}-\mathbf{A}\cdot\boldsymbol{\theta}_L\right)\\ -\frac{1}{2}\cdot\left(\mathbf{C}\cdot\boldsymbol{\theta}_L-\boldsymbol{\theta}_0\right)^T\cdot\boldsymbol{\Sigma}_0^{-1}\cdot\left(\mathbf{C}\cdot\boldsymbol{\theta}_L-\boldsymbol{\theta}_0\right)\end{bmatrix}\cdot d\boldsymbol{\theta}_L\ . \tag{5.17}$$

Given (5.17), the optimization problem in (5.8) becomes

$$\max_{\sigma_\alpha,\sigma_\beta} \quad obj = \left|\boldsymbol{\Sigma}_0\right|^{-1/2} \cdot \int_{\boldsymbol{\theta}_L} \exp\begin{bmatrix} -\frac{1}{2}\cdot\left(\log \mathbf{P}_g^{MC} - \mathbf{A}\cdot\boldsymbol{\theta}_L\right)^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \left(\log \mathbf{P}_g^{MC} - \mathbf{A}\cdot\boldsymbol{\theta}_L\right) \\ -\frac{1}{2}\cdot\left(\mathbf{C}\cdot\boldsymbol{\theta}_L - \boldsymbol{\theta}_0\right)^T \cdot \boldsymbol{\Sigma}_0^{-1} \cdot \left(\mathbf{C}\cdot\boldsymbol{\theta}_L - \boldsymbol{\theta}_0\right) \end{bmatrix} \cdot d\boldsymbol{\theta}_L \tag{5.18}$$

where $\boldsymbol{\Sigma}_0$ contains σ_α and σ_β. The objective function in (5.18) can be re-written as

$$obj = \left|\boldsymbol{\Sigma}_0\right|^{-1/2} \cdot \int_{\boldsymbol{\theta}_L} \exp\begin{Bmatrix} -\frac{1}{2}\cdot\boldsymbol{\theta}_L^T\cdot\left[\mathbf{A}^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\mathbf{A} + \mathbf{C}^T\cdot\boldsymbol{\Sigma}_0^{-1}\cdot\mathbf{C}\right]\cdot\boldsymbol{\theta}_L \\ +\left[\left(\log \mathbf{P}_g^{MC}\right)^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\mathbf{A} + \boldsymbol{\theta}_0^T\cdot\boldsymbol{\Sigma}_0^{-1}\cdot\mathbf{C}\right]\cdot\boldsymbol{\theta}_L \\ -\frac{1}{2}\cdot\left[\left(\log \mathbf{P}_g^{MC}\right)^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\log \mathbf{P}_g^{MC} + \boldsymbol{\theta}_0^T\cdot\boldsymbol{\Sigma}_0^{-1}\cdot\boldsymbol{\theta}_0\right] \end{Bmatrix} \cdot d\boldsymbol{\theta}_L \; . \tag{5.19}$$

Define

$$\mathbf{H} = \mathbf{A}^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\mathbf{A} + \mathbf{C}^T\cdot\boldsymbol{\Sigma}_0^{-1}\cdot\mathbf{C} \tag{5.20}$$

$$\mathbf{J} = \left(\log \mathbf{P}_g^{MC}\right)^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\mathbf{A} + \boldsymbol{\theta}_0^T\cdot\boldsymbol{\Sigma}_0^{-1}\cdot\mathbf{C} \tag{5.21}$$

$$\mathbf{L} = \left(\log \mathbf{P}_g^{MC}\right)^T\cdot\boldsymbol{\Sigma}_g^{-1}\cdot\log \mathbf{P}_g^{MC} + \boldsymbol{\theta}_0^T\cdot\boldsymbol{\Sigma}_0^{-1}\cdot\boldsymbol{\theta}_0 \; . \tag{5.22}$$

Given (5.20)-(5.22), Eq. (5.19) can be simplified as

$$obj = \left|\boldsymbol{\Sigma}_0\right|^{-1/2} \cdot \int_{\boldsymbol{\theta}_L} \exp\left[-\frac{1}{2}\cdot\left(\boldsymbol{\theta}_L^T\cdot\mathbf{H}\cdot\boldsymbol{\theta}_L - 2\cdot\mathbf{J}\cdot\boldsymbol{\theta}_L + \mathbf{L}\right)\right]\cdot d\boldsymbol{\theta}_L \; . \tag{5.23}$$

We further define

$$\boldsymbol{\mu}_{\boldsymbol{\theta}} = \mathbf{H}^{-1}\cdot\mathbf{J}^T \tag{5.24}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \mathbf{H}^{-1} \; . \tag{5.25}$$

Given (5.24) and (5.25), Eq. (5.23) can be re-written as

$$obj = \left(\frac{8\cdot\pi^3\cdot\left|\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\right|}{\left|\boldsymbol{\Sigma}_0\right|}\right)^{1/2} \cdot \int_{\boldsymbol{\theta}_L} \frac{1}{(2\pi)^{3/2}\cdot\left|\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\right|^{1/2}} \cdot \exp\left[-\frac{1}{2}\cdot\begin{pmatrix} \boldsymbol{\theta}_L^T\cdot\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\cdot\boldsymbol{\theta}_L - 2\cdot\boldsymbol{\mu}_{\boldsymbol{\theta}}^T\cdot\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\cdot\boldsymbol{\theta}_L + \\ \boldsymbol{\mu}_{\boldsymbol{\theta}}^T\cdot\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\cdot\boldsymbol{\mu}_{\boldsymbol{\theta}} + \mathbf{L} - \boldsymbol{\mu}_{\boldsymbol{\theta}}^T\cdot\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\cdot\boldsymbol{\mu}_{\boldsymbol{\theta}} \end{pmatrix}\right] d\boldsymbol{\theta}_L \; . \tag{5.26}$$

Eq. (5.26) can be further simplified as

$$
\begin{aligned}
obj &= \left(\frac{8 \cdot \pi^3 \cdot |\boldsymbol{\Sigma}_{\boldsymbol{\theta}}|}{|\boldsymbol{\Sigma}_0|} \right)^{1/2} \cdot \exp\left[-\frac{1}{2} \cdot \left(\mathbf{L} - \boldsymbol{\mu}_{\boldsymbol{\theta}}^T \cdot \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \cdot \boldsymbol{\mu}_{\boldsymbol{\theta}} \right) \right] \\
&\quad \cdot \int_{\boldsymbol{\theta}_L} \frac{1}{(2\pi)^{3/2} \cdot |\boldsymbol{\Sigma}_{\boldsymbol{\theta}}|^{1/2}} \cdot \exp\left[-\frac{1}{2} \cdot \left(\boldsymbol{\theta}_L - \boldsymbol{\mu}_{\boldsymbol{\theta}} \right)^T \cdot \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \cdot \left(\boldsymbol{\theta}_L - \boldsymbol{\mu}_{\boldsymbol{\theta}} \right) \right] d\boldsymbol{\theta}_L \, . \\
&= \left(\frac{8 \cdot \pi^3 \cdot |\boldsymbol{\Sigma}_{\boldsymbol{\theta}}|}{|\boldsymbol{\Sigma}_0|} \right)^{1/2} \cdot \exp\left[-\frac{1}{2} \cdot \left(\mathbf{L} - \boldsymbol{\mu}_{\boldsymbol{\theta}}^T \cdot \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \cdot \boldsymbol{\mu}_{\boldsymbol{\theta}} \right) \right]
\end{aligned}
\tag{5.27}
$$

Substituting (5.24) and (5.25) into (5.27), we have

$$
obj = \left(\frac{8 \cdot \pi^3}{|\mathbf{H}| \cdot |\boldsymbol{\Sigma}_0|} \right)^{1/2} \cdot \exp\left[-\frac{1}{2} \cdot \left(\mathbf{L} - \mathbf{J} \cdot \mathbf{H}^{-1} \cdot \mathbf{J}^T \right) \right]. \tag{5.28}
$$

Given (5.28), the optimization problem in (5.18) can be re-written as

$$
\max_{\sigma_\alpha, \sigma_\beta} \quad obj = \left(\frac{8 \cdot \pi^3}{|\mathbf{H}| \cdot |\boldsymbol{\Sigma}_0|} \right)^{1/2} \cdot \exp\left[-\frac{1}{2} \cdot \left(\mathbf{L} - \mathbf{J} \cdot \mathbf{H}^{-1} \cdot \mathbf{J}^T \right) \right]. \tag{5.29}
$$

Take logarithm on the objective function *obj* in (5.29). The optimization problem in (5.29) is cast to an equivalent optimization problem

$$
\min_{\sigma_\alpha, \sigma_\beta} \quad obj = \log|\boldsymbol{\Sigma}_0| + \log|\mathbf{H}| + \mathbf{L} - \mathbf{J} \cdot \mathbf{H}^{-1} \cdot \mathbf{J}^T \tag{5.30}
$$

where $\boldsymbol{\Sigma}_0$, **H**, **J**, and **L** are defined in (5.14), (5.20), (5.21), and (5.22) respectively. The objective function *obj* in (5.30) is not convex. In this thesis, we sweep values for σ_α and σ_β in a large range, and compute *obj* in (5.30) for every sweeping point. Based on the *obj* values at all the sweeping points, we pick up σ_α and σ_β that correspond to the minimum *obj* as the optimal σ_α and σ_β. Since computing *obj* at each sweeping point is computationally efficient, the computational cost for solving (5.30) is negligible. Once σ_α and σ_β are found, the prior distribution in (5.7) is fully determined.

5.2 Model Fitting via Bayesian Model Fusion (BMF)

Once the prior distribution in (5.7) is defined, we apply BMF [60]-[69] to solve $\boldsymbol{\theta}_L$ by MAP estimation, as shown in (5.3). Substituting (5.6)-(5.7) into (5.3) yields

$$\max_{\boldsymbol{\theta}_L} \quad pdf\left(\boldsymbol{\theta}_L \middle| \mathbf{D}\right) \propto \begin{cases} pdf\left(\alpha_L\right) \cdot pdf\left(\beta_L\right) \cdot pdf\left(\mathbf{D} \middle| \boldsymbol{\theta}_L\right) & \gamma_L \in \left[l_\gamma, u_\gamma\right] \\ 0 & \text{else} \end{cases} \tag{5.31}$$

where $\boldsymbol{\theta}_L = [\alpha_L\ \beta_L\ \gamma_L]^T$. According to (5.12) and (5.16), we observe that $pdf(\alpha_L)\cdot pdf(\beta_L)\cdot pdf(\mathbf{D} \mid \boldsymbol{\theta}_L)$ is always positive. Hence, the optimal γ_L value for the optimization problem in (5.31) must belong to $[l_\gamma\ u_\gamma]$, and the optimization problem in (5.31) is then cast to an equivalent problem

$$\begin{aligned} &\max_{\boldsymbol{\theta}_L} \quad pdf\left(\alpha_L\right) \cdot pdf\left(\beta_L\right) \cdot pdf\left(\mathbf{D} \middle| \boldsymbol{\theta}_L\right) \\ &s.t. \quad \gamma_L \in \left[l_\gamma, u_\gamma\right] \end{aligned} . \tag{5.32}$$

Substituting (5.12) and (5.16) into (5.32), we have

$$\begin{aligned} &\max_{\boldsymbol{\theta}_L} \quad \frac{1}{\left(2\pi\right)^{Q/2+1} \cdot \left|\boldsymbol{\Sigma}_0\right|^{1/2} \cdot \left|\boldsymbol{\Sigma}_g\right|^{1/2}} \cdot \exp\left[\begin{array}{c} -\frac{1}{2} \cdot \left(\mathbf{C}\cdot\boldsymbol{\theta}_L - \boldsymbol{\theta}_0\right)^T \cdot \boldsymbol{\Sigma}_0^{-1} \cdot \left(\mathbf{C}\cdot\boldsymbol{\theta}_L - \boldsymbol{\theta}_0\right) \\ -\frac{1}{2} \cdot \left(\mathbf{A}\cdot\boldsymbol{\theta}_L - \log\mathbf{P}_g^{MC}\right)^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \left(\mathbf{A}\cdot\boldsymbol{\theta}_L - \log\mathbf{P}_g^{MC}\right) \end{array}\right] . \\ &s.t. \quad \gamma_L \in \left[l_\gamma, u_\gamma\right] \end{aligned} \tag{5.33}$$

Eq. (5.33) can be re-written as

$$\begin{aligned} &\min_{\boldsymbol{\theta}_L} \quad \left(\mathbf{C}\cdot\boldsymbol{\theta}_L - \boldsymbol{\theta}_0\right)^T \cdot \boldsymbol{\Sigma}_0^{-1} \cdot \left(\mathbf{C}\cdot\boldsymbol{\theta}_L - \boldsymbol{\theta}_0\right) + \left(\mathbf{A}\cdot\boldsymbol{\theta}_L - \log\mathbf{P}_g^{MC}\right)^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \left(\mathbf{A}\cdot\boldsymbol{\theta}_L - \log\mathbf{P}_g^{MC}\right) \\ &s.t. \quad \gamma_L \in \left[l_\gamma, u_\gamma\right] \end{aligned} . \tag{5.34}$$

In this work, we set l_γ to a sufficiently small number and u_γ to a sufficiently large number. Because of this, we assume that the constraint in (5.34) is always met. The constrained optimization problem in (5.34) is then simplified as an unconstrained optimization problem

$$\min_{\boldsymbol{\theta}_L} \quad \left(\mathbf{C}\cdot\boldsymbol{\theta}_L - \boldsymbol{\theta}_0\right)^T \cdot \boldsymbol{\Sigma}_0^{-1} \cdot \left(\mathbf{C}\cdot\boldsymbol{\theta}_L - \boldsymbol{\theta}_0\right) + \left(\mathbf{A}\cdot\boldsymbol{\theta}_L - \log\mathbf{P}_g^{MC}\right)^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \left(\mathbf{A}\cdot\boldsymbol{\theta}_L - \log\mathbf{P}_g^{MC}\right). \tag{5.35}$$

It is straightforward to prove that the cost function in (5.35) is convex [73]. Hence, its global optimum can be directly solved by applying the first-order optimality condition [73]

$$\boldsymbol{\theta}_L^{MAP} = \left(\mathbf{A}^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \mathbf{A} + \mathbf{C}^T \cdot \boldsymbol{\Sigma}_0^{-1} \cdot \mathbf{C}\right)^{-1} \cdot \left(\mathbf{A}^T \cdot \boldsymbol{\Sigma}_g^{-1} \cdot \log\mathbf{P}_g^{MC} + \mathbf{C}^T \cdot \boldsymbol{\Sigma}_0^{-1} \cdot \boldsymbol{\theta}_0\right) \tag{5.36}$$

where $\log\mathbf{P}_g^{MC}$, $\boldsymbol{\Sigma}_g$, $\mathbf{A}$, $\boldsymbol{\theta}_0$, $\boldsymbol{\Sigma}_0$, $\mathbf{C}$ are defined in (4.45), (4.48), (4.50), (5.13), (5.14), and (5.15) respectively.

Once the MAP solution $\boldsymbol{\theta}_L^{MAP} = [\alpha_L^{MAP}\ \beta_L^{MAP}\ \gamma_L^{MAP}]^T$ for the late design is obtained, the rare failure rate P_f is calculated as

$$P_f^{BSSS} = \exp\left(\alpha_L^{MAP} + \gamma_L^{MAP}\right). \tag{5.37}$$

5.3 Confidence Interval Estimation

Similar to SSS, we apply bootstrap [70] to accurately estimate the confidence interval of the proposed BSSS estimator in (5.37). The key idea of bootstrap is to re-generate a large number of datasets $\{\mathbf{D}^{(n)}; n = 1, 2, \cdots, N^{BOOT}\}$ based on a statistical model without running additional transistor-level simulations. These datasets $\{\mathbf{D}^{(n)}; n = 1, 2, \cdots, N^{BOOT}\}$ are then used to repeatedly run BSSS for N^{BOOT} times and we get N^{BOOT} different failure rates $\{P_f^{BSSS(n)}; n = 1, 2, \cdots, N^{BOOT}\}$. Based on $\{P_f^{BSSS(n)}; n = 1, 2, \cdots, N^{BOOT}\}$, the statistics (hence, the confidence interval) of the estimator P_f^{BSSS} can be accurately estimated. More details about the bootstrap approach can be found in Section 4.3.2.

5.4 Algorithm Summary

Algorithm 5.1 Bayesian Scaled-Sigma Sampling (BSSS)

1. Start from the model coefficients $\boldsymbol{\theta}_E$ for the early design.
2. Given the total number of simulations N, and the number of scaling factors Q.
3. Select a set of scaling factors $\{s_q; q = 1, 2, \cdots, Q\}$ and determine the number of samples $\{N_q; q = 1, 2, \cdots, Q\}$ that we intend to generate for $\{s_q; q = 1, 2, \cdots, Q\}$ by using the approach mentioned in Section 4.3.3.
4. For each scaling factor s_q where $q \in \{1, 2, \cdots, Q\}$, sample the scaled PDF $g(\mathbf{x})$ in (4.1) for Gaussian distribution or (4.31) for Gaussian-Uniform distribution by setting $s = s_q$, and generate N_q scaled random samples by running transistor-level simulations.
5. Calculate the scaled failure rates $\{P_{g,q}^{MC}; q = 1, 2, \cdots, Q\}$ by (4.40) based on the simulation results collected in Step 4.
6. For each estimator $P_{g,q}^{MC}$ where $q \in \{1, 2, \cdots, Q\}$, calculate its variance $v_{g,q}^{MC}$ by (4.41).
7. Form the Q-dimensional vector $\log\mathbf{P}_g^{MC}$ by taking the logarithm for the estimated failure rates $\{P_{g,q}^{MC}; q = 1, 2, \cdots, Q\}$, as shown in (4.45).
8. Form the diagonal matrix $\boldsymbol{\Sigma}_g$ in (4.48) and the matrix $\mathbf{A}$ in (4.50).

9. Determine the optimal σ_α and σ_β by solving the optimization problem in (5.30), where $\mathbf{\Sigma}_0$, $\mathbf{H}$, $\mathbf{J}$, and $\mathbf{L}$ are defined in (5.14), (5.20), (5.21), and (5.22) respectively.
10. Form the prior distribution for $\boldsymbol{\theta}_L$ by using (5.4)-(5.7).
11. Calculate the MAP solution $\boldsymbol{\theta}_L$ based on (5.36), where the vector $\boldsymbol{\theta}_L$ is composed of the model coefficients α, β and γ.
12. Approximate the failure rate P_f by the estimator P_f^{BSSS} in (5.37).
13. For each estimator $P_{g,q}^{MC}$ where $q \in \{1, 2, \cdots, Q\}$, re-sample the Gaussian distribution in (4.42) for which the actual mean $P_{g,q}$ is approximated by its estimated value $P_{g,q}^{MC}$, and generate N_{RS} re-sampled values $\{P_{g,q}^{MC(n)}; n = 1, 2, \cdots, N_{RS}\}$.
14. For each data set $\{P_{g,q}^{MC(n)}; q = 1, 2, \cdots, Q\}$ where $n \in \{1, 2, \cdots, N_{RS}\}$, repeat Step 6~12 to calculate the failure rate $P_f^{BSSS(n)}$.
15. Based on the data set $\{P_f^{BSSS(n)}; n = 1, 2, \cdots, N_{RS}\}$, estimate the confidence interval of the estimator P_f^{BSSS}.

Algorithm 5.1 summarizes the simplified flow of the proposed BSSS approach. It assumes that the model coefficients $\boldsymbol{\theta}_E$ for the early design are already given, and our objective is to estimate the rare failure rate for a late design. If the prior knowledge encoded by $\boldsymbol{\theta}_E$ is accurate, BSSS is expected to offer superior accuracy over the proposed SSS method, as will be demonstrated by the numerical example in the next section.

5.5 Experimental Results

In this section, the sense amplifier shown in Figure 4-4 is used to demonstrate the efficacy of the proposed BSSS method. As mentioned in 4.4.1, the *BL* and *BLB* voltages are initially set to 1.1V and 1.2V respectively. If the output of SA is 0, the SA is considered as “PASS”. Otherwise, it is considered as “FAIL”. In this SA example, we have 45 transistors and each transistor consists of a number of multipliers. To consider process variations, 4 independent Normal random variables are used to model the random mismatch of each multiplier in the process design kit.

For testing and comparison purposes, four different approaches are implemented: (i) the brute-force

Monte Carlo approach, (ii) the minimum-norm importance sampling (MNIS), (iii) the proposed SSS method, and (iv) the proposed BSSS method. The brute-force Monte Carlo approach is used to generate the "golden" failure rate so that the accuracy of the other three methods can be quantitatively evaluated. The implementation of MNIS consists of two stages. In the first stage, 2000 transistor-level simulations are used to search the variation space and find the failure event that is most likely to occur. Next, importance sampling is applied with a shifted Normal distribution to estimate the rare failure rate. On the other hand, when implementing the proposed SSS and BSSS methods, 5 different scaling factors are empirically chosen to estimate the failure rate and 200 re-sampled data points are generated to estimate the confidence interval (i.e., Q = 5 and N_{RS} = 200) in Algorithm 4.3 and Algorithm 5.1.

We start from an initial SA design, which has 536 random variables in total. SSS is performed with 10^4 transistor-level simulations to estimate the failure rate for the 1st SA design. The estimated failure rate by SSS is 7.3×10^{-5}, which is relatively high and does not meet our specification. To verify the accuracy of SSS, we run the brute-force Monte Carlo approach with 10^6 random samples, and the estimated failure rate is 6×10^{-5}, which is considered as the "golden" failure rate for the 1st SA design. The estimated failure rate by SSS is very close to the "golden" failure rate estimated by the brute-force Monte Carlo approach, which again demonstrates the accuracy of the proposed SSS method.

Since the estimated failure rate for the 1st SA design is relatively large, we need to further improve the SA design to reduce its failure rate. By tuning the transistor sizes (i.e., increasing the number of multipliers), we obtain a 2nd SA design, which has 552 independent Normal random variables in total.

For the 2nd SA design, we first run MC with 3.5×10^6 random samples. The estimated failure rate is 7.1×10^{-6}, which is considered as the "golden" failure rate of the 2nd design. Next, we run MNIS, SSS, and BSSS with different numbers of simulations. The model coefficients for the 1st SA design learned by SSS are considered as the prior knowledge for BSSS. The estimated failure rates P_f and the 95% confidence intervals [P_f^L P_f^U] by these three approaches are shown in Table 5-1.

Table 5-1. Failure rate P_f and 95% CI [P_f^L, P_f^U] estimated by MNIS, SSS and BSSS ("golden" failure rate = 7.1×10^{-6})

# of Sims		6000	7000	8000	9000	10000
MNIS	P_f^L	0	0	0	0	0
	P_f	1.5×10^{-15}	1.2×10^{-15}	1.1×10^{-13}	4.1×10^{-13}	3.6×10^{-13}
	P_f^U	3.1×10^{-15}	2.5×10^{-15}	3.1×10^{-13}	1.1×10^{-12}	9.2×10^{-13}
SSS	P_f^L	9.0×10^{-7}	9.4×10^{-7}	6.2×10^{-7}	4.5×10^{-7}	5.1×10^{-7}
	P_f	4.4×10^{-5}	3.8×10^{-5}	2.0×10^{-5}	1.1×10^{-5}	1.0×10^{-5}
	P_f^U	2.3×10^{-4}	2.0×10^{-4}	1.7×10^{-4}	8.8×10^{-5}	7.9×10^{-5}
BSSS	P_f^L	2.4×10^{-6}	1.8×10^{-6}	1.7×10^{-6}	1.5×10^{-6}	1.5×10^{-6}
	P_f	1.3×10^{-5}	9.0×10^{-6}	8.0×10^{-6}	7.0×10^{-6}	6.2×10^{-6}
	P_f^U	4.8×10^{-5}	4.1×10^{-5}	2.8×10^{-5}	2.4×10^{-5}	2.0×10^{-5}

Studying Table 5-1, we have several observations. First, MNIS does not predict the failure rate or the 95% confidence interval accurately even with 10^4 simulations. We believe that MNIS cannot find a good “distorted” distribution in this high-dimensional example and, hence, misses the most important failure region. Second, both SSS and BSSS estimate the failure rate and the 95% confidence interval more accurately than MNIS.

Last, but more importantly, BSSS achieves significantly enhanced accuracy over SSS. From Table 5-1, we can observe that the 95% confidence interval of BSSS with 6000 simulations is substantially narrower than that of SSS with 10^4 simulations. Alternatively speaking, our proposed BSSS approach achieves more than 1.7× runtime speedup over SSS in this example.

Chapter 6

Thesis Summary & Future Work

6.1 Summary

With aggressive technology scaling, process variation has become a growing concern for today's integrated circuits (ICs). Due to large-scale process variations, what we obtain after fabrication can be quite different from what we design. If the deviation is significant, the functionality of the circuit could differ a lot from what we expect, and the circuit may fail to work. SRAM has been widely embedded in a large amount of semiconductor chips and, hence, designing a robust SRAM with sufficiently high production yield under large-scale process variations is an important task for IC design community.

SRAM typically contains a large number of replicated circuit components (e.g., SRAM bit-cell, SRAM column, sense amplifier, etc.). To achieve sufficiently high yield, the failure event of each circuit component must be extremely rare. For instance, the failure rate of an SRAM bit-cell must be less than 10^{-8}~10^{-6} so that the full microprocessor system, containing millions of SRAM bit-cells, can achieve sufficiently high yield. For this reason, efficient approaches for estimating the rare circuit failure events are highly desired for SRAM design.

A number of statistical approaches have been developed in the literature [24]-[45]. Most of these traditional methods focus on failure rate estimation for SRAM bit-cells that consist of few (e.g., 6~10) transistors. In these cases, only a small number of (e.g., 6~20) independent random variables are used to model process variations and, hence, the corresponding variation space is low-dimensional. Rare failure event analysis in a high-dimensional space has become more and more important. Unfortunately, most of these traditional approaches [24]-[45] cannot be efficiently applied. To address this issue, we propose three novel approaches to estimate the rare failure events for SRAM circuits in a high-dimensional space in this thesis.

First, we propose a subset simulation (SUS) approach in Chapter 3 to estimate the rare failure events for circuits that have continuous performances of interest. The key idea of SUS is to express the rare failure probability as the product of several large conditional probabilities by introducing a number of intermediate failure events. A Markov chain Monte Carlo algorithm (i.e., the modified Metropolis algorithm) is used to accurately estimate the intermediate conditional probabilities and, eventually, the rare failure rate of a given circuit. In addition, a statistical methodology is further developed to reliably estimate the confidence interval of SUS. An SRAM column example designed in nanoscale technologies demonstrate that SUS offers superior estimation accuracy over the traditional importance sampling technique (i.e., MNIS) when hundreds of random variables are used to model process variations.

Second, we propose a scaled-sigma sampling (SSS) approach in Chapter 4 to estimate the rare failure events for circuits that have discrete performances of interest. The proposed SSS approach is based upon an analytical model derived from the theorem of "soft maximum". It is statistically formulated as a regression modeling problem and optimally solved by MLE. To quantitatively assess the accuracy of SSS, the confidence interval is estimated by bootstrap for our proposed SSS estimator. Two circuit examples demonstrate the efficacy of our proposed SSS approach in a high-dimensional space.

Third, to further reduce the simulation cost, we propose a Bayesian scaled-sigma sampling (BSSS) approach in Chapter 5, which can be considered as an extension of SSS. BSSS explores the "similarity" between different SSS models fitted at different design stages, and encodes it as our prior knowledge. Next, the SSS model is fitted by using MAP estimation with consideration of the prior knowledge. Experimental results demonstrate that the proposed BSSS approach achieves superior accuracy over MNIS and SSS when the dimensionality of the variation space is more than a few hundred.

6.2 Future Work

There are a number of research directions that we could further explore in the future.

First, we use an extremely conservative upper bound to approximate the SRAM column failure rate in Section 3.5. How to find a better approximation of the SRAM column failure rate without running intensive transistor-level simulations is still an open question. One possible solution relies on inclusion-exclusion principle [75]. Namely, we first estimate the probability that a single SRAM bit-cell fails, the probability that two SRAM bit-cells fail simultaneously, the probability that three SRAM bit-cells fail simultaneously, etc. Based on these probability values, we approximate the SRAM column failure rate by applying inclusion-exclusion principle.

Second, we apply a heuristic approach to determine the scaling factors and allocate samples for these scaling factors in Section 4.3.3. This heuristic approach is based on several assumptions and user pre-defined parameters. These assumptions and empirically chosen parameters cannot guarantee an optimal solution. Selecting an appropriate set of scaling factors is crucial for SSS. Therefore, if we have a better way to determine the scaling factors, we can further increase the efficiency of SSS. One interesting future work is to develop better approaches to determine the scaling factors. Possible solutions may formulate and solve optimization problems.

Third, we present several scenarios where SSS may fail to work in Section 4.5. We can detect and fix such failures in several cases. However, there are still some cases where we cannot fix or even detect the failures. One future research direction is to develop feasible solutions to monitor the SSS failures.

Last, we define α_L and β_L as two Normal random variables and γ_L as a uniform random variable in Section 5.1. Namely, we constrain α_L and β_L to be close to α_E and β_E respectively, and we do not utilize any information from γ_E. There are other possible ways to define the prior distribution for the three model coefficients $[\alpha_L\ \beta_L\ \gamma_L]$. Another interesting research work is to explore other distributions to fully utilize our prior knowledge about the model coefficients. Generally speaking, if we use more information about the model coefficients when we define the prior distribution, we can fit the SSS model by the BMF technique more efficiently.

Bibliography

[1] E. Seevinck, F. J. List and J. Lohstroh, “Static-noise margin analysis of MOS SRAM cells,” *IEEE JSSC*, vol. 22, no. 5, pp. 748-754, Oct. 1987.

[2] A. J. Bhavnagarwala, X. Tang and J. D. Meindl, “The impact of intrinsic device fluctuations on CMOS SRAM cell stability,” *IEEE JSSC*, vol. 36, no. 4, pp. 658-665, Apr. 2001.

[3] R. Heald and P. Wang, “Variability in sub-100nm SRAM designs,” *IEEE ICCAD*, pp. 347-352, 2004.

[4] R. V. Joshi, S. Mukhopadhyay, D. W. Plass, Y. H. Chan, C. Chuang and A. Devgan, “Variability analysis for sub-100 nm PD/SOI CMOS SRAM cell,” *IEEE ESSCIRC*, pp. 211-214, Sep. 2004.

[5] S. Mukhopadhyay, H. Mahmoodi and K. Roy, “Statistical design and optimization of SRAM cell for yield enhancement,” *IEEE ICCAD*, pp. 10-13, Nov. 2004.

[6] L. Chang, D. M. Fried, J. Hergenrother, J. W. Sleight, R. H. Dennard, R. K. Montoye, L. Sekaric, S. J. McNab, A. W. Topol, C. D. Adams, K. W. Guarini and W. Haensch, “Stable SRAM cell design for the 32 nm node and beyond,” *Symp. VLSI Technology Dig.*, pp. 128–129, June 2005.

[7] A. Bhavnagarwala, S. Kosonocky, C. Radens, K. Stawiasz, R. Mann, Q. Ye and K. Chin, “Fluctuation limits & scaling opportunities for CMOS SRAM cells,” *IEEE IEDM*, pp. 659-662, Dec. 2005.

[8] K. Takeda, H. Ikeda, Y. Hagihara, M. Nomura and H. Kobatake, “Redefinition of write Margin for next-generation SRAM and write-margin monitoring circuit,” *IEEE ISSCC*, Feb. 2006.

[9] K. Agarwal and S. Nassif, “Statistical analysis of SRAM cell stability,” *IEEE DAC*, pp. 57-62, 2006.

[10] B. H. Calhoun and A. P. Chandrakasan, “Static noise margin variation for sub-threshold SRAM in 65-nm CMOS,” *IEEE JSSC*, vol. 41, no. 7, pp. 1673-1679, July 2006.

[11] B. H. Calhoun and A. P. Chandrakasan, “A 256-kb 65-nm sub-threshold SRAM design for ultra-low-voltage operation,” *IEEE JSSC*, vol. 42, no. 3, pp. 680-688, Mar. 2007.

[12] B. H. Calhoun, Y. Cao, X. Li, K. Mai, L. T. Pileggi, R. A. Rutenbar and K. L. Shepard, "Digital circuit design challenges and opportunities in the era of nanoscale CMOS," *Proc. IEEE*, vol. 96, no. 2, pp. 343-365, Feb. 2008.

[13] M. H. Abu-Rahma, K. Chowdhury, J. Wang, Z. Chen, S. S. Yoon and M. Anis, "A methodology for statistical estimation of read access yield in SRAMs," *IEEE DAC*, pp. 205-210, 2008.

[14] J. Wang, S. Nalam and B. H. Calhoun, "Analyzing static and dynamic write margin for nanometer SRAMs," *IEEE ISLPED*, pp. 129-134, Aug. 2008.

[15] H. S. Yang, et al., "Scaling of 32nm low power SRAM with high-k metal gate," *IEEE IEDM*, Dec. 2008.

[16] S. Yaldiz, U. Arslan, X. Li and L. Pileggi, "Efficient statistical analysis of read timing failures in SRAM circuits," *IEEE ISQED*, pp. 617-621, Mar. 2009.

[17] Y. Wang, U. Bhattacharya, F. Hamzaoglu, P. Kolar, Y. Ng, L. Wei, Y. Zhang, K. Zhang and M. Bohr, "A 4.0 GHz 291 Mb voltage-scalable SRAM design in a 32 nm high-k + metal-gate CMOS technology with integrated power management," *IEEE JSSC*, vol. 45, no. 1, pp. 103-110, Jan. 2010.

[18] G. Chen, D. Sylvester, D. Blaauw and T. Mudge, "Yield-driven near-threshold SRAM design," *IEEE TVLSI*, vol. 18, no. 11, pp. 1590-1598, Nov. 2010.

[19] H. Sun, J. Zhao, F. Wang, N. Zheng and T. Zhang, "Cost-efficient built-in repair analysis for embedded memories with on-chip ECC," *IEEE ISAS*, pp. 95-100, June 2011.

[20] H. Pilo, I. Arsovsi, K. Batson, G. Braceras, J. Gabric, R. Houle, S. Lamphier, C. Radens and A. Seferagic, "A 64 Mb SRAM in 32 nm high-k metal-gate SOI technology with 0.7 V operation enabled by stability, write-ability and read-ability enhancements," *IEEE JSSC*, vol. 47, no. 1, pp. 97-106, Jan. 2012.

[21] B. Rooseleer, S. Cosemans and W. Dehaene, "A 65 nm, 850 MHz, 256 kbit, 4.3 pJ/access, ultra low leakage power memory using dynamic cell stability and a dual swing data link," *IEEE JSSC*, vol. 47, no. 7, pp. 1784-1796, July 2012.

[22] S. Lu, H. Huang, J. Huang and P. Ning, "Synergistic reliability and yield enhancement techniques for embedded SRAMs," *IEEE TCAD*, vol. 32, no. 1, pp. 165-169, Jan. 2013.

[23] W. Zhang, "IC spatial variation modeling: algorithms and applications," Ph.D. dissertation, Dept.

Elect. Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, 2012.

[24] D. E. Hocevar, M. R. Lightner and T. N. Trick, "A study of variance reduction techniques for estimating circuit yields," *IEEE TCAD*, vol. 2, no. 3, pp. 180-192, July 1983.

[25] T. C. Hesterberg, "Advances in importance sampling," Ph.D. dissertation, Dept. Statistics, Stanford Univ., Stanford, CA, 1988.

[26] R. Kanj, R. Joshi and S. Nassif, "Mixture importance sampling and its application to the analysis of SRAM designs in the presence of rare failure events," *IEEE DAC*, pp. 69-72, 2006.

[27] R. O. Topaloglu, "Early, accurate and fast yield estimation through Monte Carlo-alternative probabilistic behavioral analog system simulations," *IEEE VLSI Test Symp.*, pp. 137-142, 2006.

[28] S. Srivastava and J. Roychowdhury, "Rapid estimation of the probability of SRAM failure due to MOS threshold variations," *IEEE CICC*, 2007.

[29] C. Gu and J. Roychowdhury, "An efficient, fully nonlinear, variability aware non-Monte-Carlo yield estimation procedure with applications to SRAM cells and ring oscillators," *IEEE ASP-DAC*, pp. 754-761, 2008.

[30] V. Veetil, D. Sylvester and D. Blaauw, "Efficient Monte Carlo based incremental statistical timing analysis," *IEEE DAC*, pp. 676-681, 2008.

[31] L. Dolecek, M. Qazi, D. Shah and A. Chandrakasan, "Breaking the simulation barrier: SRAM evaluation through norm minimization," *IEEE ICCAD*, pp. 322-329, 2008.

[32] J. Jaffari and M. Anis, "On efficient Monte Carlo-based statistical static timing analysis of digital circuits," *IEEE ICCAD*, pp. 196-203, Nov. 2008.

[33] J. Wang, S. Yaldiz, X. Li and L. T. Pileggi, "SRAM parametric failure analysis," *IEEE DAC*, pp. 496-501, 2009.

[34] A. Singhee and R. A. Rutenbar, "Statistical blockade: very fast statistical simulation and modeling of rare circuit events, and its application to memory design," *IEEE TCAD*, vol. 28, no. 8, pp. 1176-1189, Aug. 2009.

[35] J. Jaffari and M. Anis, "Adaptive sampling for efficient failure probability analysis of SRAM cells," *IEEE ICCAD*, pp. 623- 630, 2009.

[36] J. Jaffari and M. Anis, "Correlation controlled sampling for efficient variability analysis of analog

circuits," *IEEE DATE*, pp. 1305-1308, Mar. 2010.

[37] M. Qazi, M. Tikekar, L. Dolecek, D. Shah and A. Chandrakasan, "Loop flattening and spherical sampling: highly efficient model reduction techniques for SRAM yield analysis," *IEEE DATE*, pp. 801-806, 2010.

[38] R. A. Fonseca, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel and N. Badereddine, "A statistical simulation method for reliability analysis of SRAM core-cells," *IEEE DAC*, pp. 853-856, 2010.

[39] K. Katayama, S. Hagiwara, H. Tsutsui, H. Ochi and T. Sato, "Sequential importance sampling for low-probability and high-dimensional SRAM yield analysis," *IEEE ICCAD*, pp. 703-708, 2010.

[40] J. Jaffari and M. Anis, "On efficient LHS-based yield analysis of analog circuits," *IEEE TCAD*, vol. 30, no. 1, pp. 159-163, Jan. 2011.

[41] R. Kanj and R. Joshi, "A novel sample reuse methodology for fast statistical simulations with applications to manufacturing variability," *IEEE ISQED*, pp. 672-678, Mar. 2012.

[42] R. Kanj, R. Joshi, Z. Li, J. Hayes and S. Nassif, "Yield estimation via multi-cones," *IEEE DAC*, pp. 1107-1112, 2012.

[43] C. Kuo, W. Hu, Y. Chen, J. Kuan and Y. Cheng, "Efficient trimmed-sample Monte Carlo methodology and yield-aware design flow for analog circuits," *IEEE DAC*, pp. 1113-1118, 2012.

[44] F. Gong, "Stochastic modeling and analysis of custom integrated circuits," Ph.D. dissertation, Dept. Elect. Eng., Univ. California Los Angeles, CA, 2012.

[45] S. Sun, Y. Feng, C. Dong and X. Li, "Efficient SRAM failure rate prediction via Gibbs sampling," *IEEE TCAD*, vol. 31, no. 12, pp. 1831-1844, Dec. 2012.

[46] F. Wood and T. L. Griffiths, "Particle filtering for nonparametric Bayesian matrix factorization," *Advances in Neural Information Processing Systems*, vol. 19, pp. 1513–1520, 2007.

[47] G. Grisetti, C. Stachniss and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Trans. Robotics*, vol. 23, no. 1, pp. 34-46, Feb. 2007.

[48] T. Bengtsson, P. Bickel and B. Li, "Curse-of-dimensionality revisited: collapse of the particle filter in very large scale systems," *Probability and Statistics: Essays in Honor of David A. Freedman,* D. Nolan and T. Speed, Eds., vol. 2, Institute of Mathematical Statistics, pp. 316–334, 2008.

[49] R. Levy, F. Reali and T. L. Griffiths, “Modeling the effects of memory on human online sentence processing with particle filters,” *Advances in Neural Information Processing Systems*, vol. 21, pp. 937–944, 2008.

[50] D. Törnqvist, T. B. Schön, R. Karlsson and F. Gustafsson, “Particle filter SLAM with high dimensional vehicle model,” *J. Intelligent Robotic Systems*, vol. 55, no. 4-5, pp. 249–266, Aug. 2009.

[51] A. Doucet and A. M. Johansen, “A tutorial on particle filtering and smoothing: fifteen years later,” in *Handbook of Nonlinear Filtering*. Cambridge: Cambridge University Press, 2009.

[52] J. L. Austerweil and T. L. Griffiths, “A nonparametric Bayesian framework for constructing flexible feature representations,” *Psychological Review*, vol. 120, no. 4, pp. 817–851, 2013.

[53] P. Glasserman, P. Heidelberger, P. Shahabuddin and T. Zajic, “Multilevel splitting for estimating rare event probabilities,” *J. Operations Research*, vol. 47, no. 4, pp 585-600, Apr. 1999.

[54] S. Au and J. L. Beck, “Estimation of small failure probabilities in high dimensions by subset simulation,” *Probabilistic Eng. Mech.*, vol. 16, no. 4, pp. 263-277, Oct. 2001.

[55] S. Au and J. L. Beck, “Subset simulation and its application to seismic risk based on dynamic analysis,” *J. Eng. Mech.*, vol. 129, no. 8, pp. 901-917, Aug. 2003.

[56] P. S. Koutsourelakis, H. J. Pradlwarter and G. I. Schuëller, “Reliability of structures in high dimensions, part I: algorithms and applications,” *Probabilistic Eng. Mech.*, vol. 19, no. 4, pp. 409-417, Oct. 2004.

[57] G. I. Schuëller, H. J. Pradlwarter and P. S. Koutsourelakis, “A critical appraisal of reliability estimation procedures for high dimensions,” *Probabilistic Eng. Mech.*, vol. 19, no. 4, pp. 463-474, Oct. 2004.

[58] A. Guyader, N. Hengartner and E. Matzner-Løber, “Simulation and estimation of extreme quantiles and extreme probabilities,” *Appl. Math. Optimization*, vol. 64, no. 2, pp. 171-196, Oct. 2011.

[59] F. Cérou, P. D. Moral, T. Furon and A. Guyader, “Sequential Monte Carlo for rare event estimation,” *Stat. Computing*, vol. 22, no. 3, pp. 795-808, May 2012.

[60] X. Li, W. Zhang, F. Wang, S. Sun and C. Gu, “Efficient parametric yield estimation of analog/mixed-signal circuits via Bayesian model fusion,” *IEEE ICCAD*, pp. 627-634, 2012.

[61] F. Wang, W. Zhang, S. Sun, X. Li and C. Gu, "Bayesian model fusion: large-scale performance modeling of analog and mixed-signal circuits by reusing early-stage data," *IEEE DAC*, 2013.

[62] X. Li, F. Wang, S. Sun and C. Gu, "Bayesian model fusion: a statistical framework for efficient pre-silicon validation and post-silicon tuning of complex analog and mixed-signal circuits," *IEEE ICCAD*, pp. 795-802, 2013.

[63] C. Fang, F. Yang, X. Zeng and X. Li, "BMF-BD: Bayesian model fusion on Bernoulli distribution for efficient yield estimation of integrated circuits," *IEEE DAC*, 2014.

[64] L. Yu, S. Saxena, C. Hess, A. Elfadel, D. Antoniadis and D. Boning, "Remembrance of transistors past: compact model parameter extraction using Bayesian inference and incomplete new measurements," *IEEE DAC*, 2014.

[65] S. Zhang, X. Li, R. D. Blanton, J. Machado da Silva, J. M. Carulli and K. M. Butler, "Bayesian model fusion: enabling test cost reduction of analog/RF circuits via wafer-level spatial variation modeling," *IEEE ITC*, 2014.

[66] S. Sun, F.Wang, S. Yaldiz, X. Li, L. Pileggi, A. Natarajan, M. Ferriss, J. Plouchart, B. Sadhu, B. Parker, A. Valdes-Garcia, M. Sanduleanu, J. Tierno and D. Friedman, "Indirect performance sensing for on-chip self-healing of analog and RF circuits," *IEEE TCAS-I*, vol. 61, no. 8, pp. 2243-2252, Aug. 2014.

[67] J. Liaperdos, H. Stratigopoulos, L. Abdallah, Y. Tsiatouhas, A. Arapoyanni and X. Li, "Fast deployment of alternate analog test using Bayesian model fusion," *IEEE DATE*, pp. 1030-1035, 2015.

[68] C. Fang, Q. Huang, F. Yang, X. Zeng, X. Li and C. Gu, "Efficient bit error rate estimation for high-speed link by Bayesian model fusion," *IEEE DATE*, pp. 1024-1029, 2015.

[69] Q. Huang, C. Fang, F. Yang, X. Zeng and X. Li, "Efficient multivariate moment estimation via Bayesian model fusion for analog and mixed-signal circuits," *IEEE DAC*, 2015.

[70] B. Efron and R. Tibshirnani, *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1993.

[71] A. Papoulis and S. Pillai, *Probability, Random Variables and Stochastic Process*. McGraw-Hill, 2001.

[72] C. Bishop, *Pattern Recognition and Machine Learning*. Prentice Hall, 2007.

[73] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2009.

[74] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, 2010.

[75] A. Björklund, T. Husfeldt and M. Koivisto. "Set partitioning via inclusion-exclusion," *SIAM J. Comput*, vol. 39, no. 2, pp. 546-563, 2009.